Torgeir Dingsøyr (

LNCS 3281

Software
Improven

11th European Conference,
Trondheim, Norway, Noven
Proceedings

Torgeir Dingsøyr (Ed.)

Software Process Improvement

11th European Conference, EuroSPI 2004
Trondheim, Norway, November 10-12, 2004
Proceedings

Springer

Volume Editor

Torgeir Dingsøyr
SINTEF ICT
S P Andersens v 15, 7465 Trondheim, Norway
E-mail: Torgeir.Dingsoyr@sintef.no

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2, K.6, K.4.2

ISSN 0302-9743
ISBN 3-540-23725-9 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11341086 06/3142 5 4 3 2 1 0

Preface

This was the first year that the European Software Process Improvement Conference – EuroSPI – had a separate research track with its own proceedings. The EuroSPI conference is in its eleventh year, and has become the main meeting place in Europe for the software industry and academia to discuss software process improvement. The conference deals with software process improvement in a broad sense, investigating organizational issues as well as methods and tools for software process improvement.

EuroSPI is an initiative financed by a consortium of Nordic research centers and user networks (SINTEF, DELTA and STTF), ASQF, a German quality assurance association, and ISCN in Ireland, the coordinating network partner.

The research papers describe innovative and significant work in software process improvement, which is relevant to the software industry. The papers are readable for a scientific and industrial audience, and support claims with appropriately described evidence or references to relevant literature.

Thirty-one papers were submitted in this year's research track, and each paper was sent to three or four members of the program committee or additional reviewers. Papers were evaluated according to originality, significance of the contribution, quality of the written and graphical presentation, research method applied, and appropriateness of comparison to relevant research and literature. Almost 100 reviews were received and 18 papers were selected for presentation in the research track, giving a rejection rate of 42%. Many high-quality submissions had to be rejected because of limited space in the conference program.

The selected papers cover a wide area in software process improvement, from improving agile development methods, techniques for software process improvement, and knowledge management in software companies to effort estimation and global software development.

I would like to thank the paper authors for providing papers of high quality, and the program committee and additional reviewers for critiques, praise and advice on how to make the papers even better.

For further information about future EuroSPI conferences, see www.eurospi.net.

August 2004 Torgeir Dingsøyr

EuroSPI 2004 Conference Organization

General Chair

Dr. Richard Messnarz, ISCN, Ireland

Local Chair

Nils Brede Moe, SINTEF ICT, Norway

Scientific Programme Committee Chair

Dr. Torgeir Dingsøyr, SINTEF ICT, Norway

Industrial Programme Committee Chairs

Jørn Johansen, DELTA, Denmark
Mads Christiansen, DELTA, Denmark
Risto Nevalainen, STTF, Finland

Industry Chair

Dr. Bernd Hindel, ASQF, Germany

Exhibition Chair

Robert Treffny, ASQF, Germany

Tutorial Chair

Dr. Richard Messnarz, ISCN, Ireland

Research Track Program Committee

Pekka Abrahamsson, VTT Electronics, Finland
Vincenzo Ambriola, University of Pisa, Italy
Aybuke Aurum, University of New South Wales, Australia
Stefan Biffl, Vienna University of Technology, Austria
Miklos Biro, University of Budapest, Hungary
Christian Bunse, Fraunhofer IESE, Germany
Marcus Ciolkowski, University of Kaiserslautern, Germany
Karl Cox, NICTA, Australia
Kevin C. Desouza, University of Illinois, USA

Taz Daughtrey, James Madison University, USA
Howard Duncan, Dublin City University, Ireland
Tore Dybå, SINTEF, Norway
Jan Pries-Heje, IT University of Copenhagen, Denmark
Natalia Juristo, Polytechnical University of Madrid, Spain
Karl Heinz Kautz, Copenhagen Business School, Denmark
Jyrki Kontio, Helsinki University of Technology, Finland
Dieter Landes, Coburg University of Applied Sciences, Germany
Mikael Lindvall, Fraunhofer Center, USA
Patricia McQuaid, California Polytechnic State University, USA
Jürgen Münch, Fraunhofer IESE, Germany
Matthias Müller, University of Karlsruhe, Germany
Markku Oivo, University of Oulu, Finland
Elixabete Ostolaza, European Software Institute, Spain
Ita Richardson, University of Limerick, Ireland
Gunther Ruhe, University of Calgary, Canada
Per Runeson, Lund Institute of Technology, Sweden
Kurt Schneider, University of Hannover, Germany
Martin Shepperd, Bournemouth University, UK
Magne Jørgensen, Simula Research Laboratory, Norway
Tor Stålhane, Norwegian University of Science and Technology, Norway
Timo Varkoi, Tampere University of Technology, Finland
Claes Wohlin, Blekinge Institute of Technology, Sweden

Additional Reviewers

Ulrike Becker-Kornstaedt, USA
M. Letizia Jaccheri, Norwegian University of Science and Technology, Norway
Daniel Karlström, Lund Institute of Technology, Sweden
Ana M. Moreno, Universidad Politecnica de Madrid, Spain
Knut H. Rolland, Norwegian University of Science and Technology, Norway
Maribel Sanchez-Segura, Universidad Carlos III de Madrid, Spain

Table of Contents

On-Site Customer in an XP Project: Empirical Results from a Case Study

Juha Koskela and Pekka Abrahamsson

VTT Technical Research Centre of Finland
P.O.Box 1100, FIN-90571 Oulu, Finland
{Juha.Koskela,Pekka.Abrahamsson}@vtt.fi

Abstract. Extreme programming (XP), similarly to other agile software development methods, values close collaboration with customers. One of the XP practices suggests that the customer should be 100% available for the development team. Anecdotal evidence suggests that the XP customer role is costly, difficult and demanding. However, very few empirical studies have been published on the role of customer in an XP project. The results of this controlled case study are in line with the common belief that the on-site customer's role is indeed demanding, requiring a strong ability to resolve issues rapidly. Yet, the study also offers contrasting findings in terms of required actual customer involvement in the development project. This empirical case demonstrates that while the customer was present close to 100% with the development team, only 21% of his work effort was required to assist the team in the development. However, it is also shown that an on-site customer may create a false sense of confidence in the system under development. The implications of these and other findings are discussed.

Keywords: Extreme programming, on-site customer, customer involvement

1 Introduction

Extreme programming (XP), first introduced in [1], is focused on generating early releases of working products and aims to deliver business value from the very beginning of the project. The role of customer is highly valued in XP, and it is considered important for the success of the project [2, 3]. The on-site customer practice of XP suggests that the customer should be 100% available for the development team, so as to be able to provide quick help in, e.g., answering questions and resolving problems.

This paper reports the empirical results from a controlled extreme programming case study, where the customer was present close to 100% of the development time. A team of four developers was acquired to implement a system for managing the research data obtained over years at a Finnish research institute. Both quantitative and qualitative data of the on-site customer role in XP is provided. The quantitative data consists of customer effort usage and effort distribution. The qualitative data comprises development diaries maintained by the developers, a customer diary, postmortem analysis session recordings and developer interviews. It has been argued that the XP customer role is demanding and that it requires lots of involvement [e.g. 4, 5-7]. While this study supports these claims, it also yields contrasting results. It is shown that the on-site customer offers the team a unique situation to consult him

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 1–11, 2004.

whenever needed. The development team perceives this as a strong demonstration of commitment to their work. The data also reveals that the on-site customer is in danger to create a false sense of confidence in the remaining of the customer organization. The results of this study offer empirical evidence to support the common belief that the on-site customer role is demanding and requires a strong ability to resolve issues rapidly. However, the empirical case demonstrates that although the customer was close to 100% present with the development team, only 21% of his work effort was required to assist the team in the development.

The paper is organized as follows. The following section introduces extreme programming and the related research. This is followed by a description of research settings, research methods and data collection methods. Section four presents the results and in section five implications of these findings are discussed. Section six concludes the paper.

2 Extreme Programming

This section introduces extreme programming and related research, focusing on the on-site customer.

Agile methods have gained a significant amount of attention in the field of software engineering in the last few years. Currently, extreme programming (XP) is the best-known agile method. XP is primarily designed for object-oriented projects using a maximum of a dozen programmers at one location [3]. This kind of situation is called “agile home ground” by Boehm [8]. The XP process is characterized by short development cycles, incremental planning, continuous feedback, and reliance on communication and evolutionary design [2]. The core of XP is made up of a simple set of common-sense practices. These practices are planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer, coding standards, open workspace and just rules. For more information about XP and an overview of other agile methods readers are referred to [e.g. 9, 10].

From the viewpoint of this study, the most interesting XP practice is the on-site customer. It has been suggested that the customer should be available for the development team throughout the project, to answer questions and resolve problems, for example. In XP, the customer is the person who sits with the project team, generates and prioritizes stories, provides acceptance tests for each release, and makes the final business decisions [11]. Therefore, the on-site customer delivers the requirements and represents all the knowledge that must be available for the development team. Despite this important role, there are only few empirically validated studies focusing on the on-site customer available.

Wallace et al. [12] list three possible customer locations: on-site customer, off-site customer and remote customer. According to XP literature [e.g. 2, 3, 13], the customer optimally works in the same room with the developers. However, this is not always possible; for example, the customer may be too valuable to be on-site [3]. According to Jeffries et al. [3], an XP project may survive even without customer presence, but the project will go faster and smoother if the customer is on-site. If the project team does not include a customer, they have to plan further in advance, which, for its part, adds the level of risk in the project [2].

Yet, it is not only the customer working on-site that makes an XP project successful. According to XP literature [e.g. 2, 11], it is also important to have a customer who plays the role well. According to Beck and Fowler [11, p. 18], a good customer understands the domain, is aware of how software can provide business value in the domain, can make decisions about what is needed at a given moment and what is needed later, and is willing to accept ultimate responsibility for the success or failure of the project. Martin et al. [4] established three research hypotheses covering the characteristics of the customer, the skills of the customer, and the location of the customer. They found the role of XP customer a very demanding one, requiring, e.g., preparation, skills, attention to detail, and the ability to make critical decisions. Martin et al. [4] report that even an ideal preparation for the customer role may not be sufficient to ensure success in the XP customer role.

Farell et al. [5] describe a successful XP implementation from the viewpoint of the customer. According to Farell et al. [5, pp. 4], "it is critical to have a high degree of customer involvement in the process." Griffin [6] has also come to similar conclusions regarding XP implementation, recommending the key customer contact(s) to be placed close to the development team. Nawrocki et al. [14, pp. 294] argue that "a close, personal contact with customers and their instant presence is a must for XP-like processes". Lippert et al. [15] have written a book in which they describe their experiences of XP practices. The authors emphasize the importance of smooth communication between development team and customer. Stephens and Rosenberg [7], for their part, provide a critical viewpoint towards XP in their book. According to Stephens and Rosenberg [7, pp. 133], "the trouble with on-site customer done the XP way is that if the on-site customer is a single person, she becomes a single point of failure in an incredibly difficult, stressful, high-profile position of great responsibility". Table 1 summarizes the most critical arguments of related research.

3 Research Design

This section describes how the research design for the study is laid out.

3.1 Research Setting

A team of four developers was set up to implement an intranet application (called eXpert) for managing the research data obtained over years at a Finnish research institute. The four developers were 5th to 6th year university students, all with 1 to 4 years of industrial experience in software development. The team members were well-versed in the Java programming language and object-oriented analysis and design approaches. Two weeks prior to project launch, the team performed a self-study session by studying two basic books on XP [i.e., 2, 3]. A two-day hands-on training on XP practices, development environment and software configuration management (SCM) was organized to ensure that the team had a basic understanding on XP issues and the specific technical environment used. As development environment, an Eclipse integration framework (http://www.eclipse.org) was used, which is an open source initiative supported by major software engineering tool manufacturers. CVS (Concurrent Versions System) was used as project SCM tool and the JUnit testing framework for unit testing. Both CVS client and JUnit are integrated as a default in the Eclipse

Table 1. Summary of related research

Claim, argument or suggestion	Description	References
A high degree of customer involvement is required	It is critical to have a high degree of customer involvement in the process	[5, 6, 14]
The role of on-site customer is very demanding	XP customer role is highly demanding, requiring, e.g., preparation, skills, attention to detail, and ability to make critical decisions	[4, 7]
The customer should work in the same room with developers	Optimum conditions for an XP project are provided by the customer sharing the same workspace as developers	[2-4]

environment. The application was written in Java and JSP (JavaServer Pages) and it used the MySQL relational database in storing link data. In addition, the Apache Tomcat 4 Servlet/JSP container was used because of its implementation of the JSP 1.2 specifications of Java Software.

The team worked in a co-located development environment. The customer (i.e., the first author) shared the same office space with the development team. The office space and workstations were organized according to the suggestions made in the XP literature to support efficient teamwork.

3.2 Research Method

A detailed description of the general research method, i.e., the controlled case study approach, utilized in this study can be found in [16]. The controlled case study approach strives for replication (experimentation) and in-depth data collection (case study) and it also has the ability to change the process (action research) in a close-to-industry setting in which also business pressure is present [16]. The first author was in the role of on-site customer, participating in planning game, acceptance testing, post-mortem analysis, project meetings and coaching activities. On average, he spent over 80% of his work time in the same room with the developers. The second author was acting in the role of management in the study, mediating the release post-mortem analysis [17] sessions, which were performed after each software release. These post-mortem analysis sessions served as a process change mechanism, where the project team could propose changes to the implementation process.

3.3 Data Collection

Both quantitative and qualitative data were collected. Quantitative data consisted of customer effort usage and effort distribution. Qualitative data included development diaries maintained by the developers, a customer diary, post-mortem analysis session recordings and developer interviews. The developers and the customer were updating their diaries continuously during the project, tracking time and filling in observations. As indicated by XP principles [2], the customer organization placed explicit value on

data collection, thus ensuring alignment with agile software development principles (http://www.agilemanifesto.org).

4 Results

This section presents the results of the study, including both quantitative and qualitative data concerning the on-site customer role in XP. Table 2 provides basic information about the size and schedule of the eXpert project. The system development was carried out in six iterations, of which the first three took up two weeks of calendar time, the next two one week, while the sixth iteration was a two-day correction release. The developers were mainly working six hours a day for four days a week. Detailed data of the eXpert project can be found in [18].

Table 2. Background information of the eXpert project

Collected data	R1	R2	R3	R4	R5	R6	Total
Calendar time (weeks)	2	2	2	1	1	0.4	8.4
Total work effort (h)	195	190	192	111	96	36	820
# User stories implemented	5	9	9	4	3	4	34
# Tasks defined	10	30	18	21	19	9	107

4.1 Customer Effort Usage and Distribution

Figure 1 shows the customer presence for each release (i.e. the time the customer was spending in the same room with the developers). As it can be seen, the customer was present at an average of 83%. Figure 1 also reveals that customer presence was at its highest in the first iteration, while decreasing to iteration three, and increasing to the average level in the forth iteration. The change of course can be explained by the change in iteration length from two weeks to one week. When the customer saw that everything was happening at a faster pace, he was trying to be present as much as possible. In the third iteration, the customer presence was at its highest in hours (59 hours), but lowest in percentage (72%). This results from a fragmented presence of the developers during the third iteration. At that time, the developers were working at more differing times compared to the first two iterations, for example.

Figure 2 shows the actual effort the customer spent in performing project activities in each release (i.e. the time the customer spent in XP activities). Despite the high customer presence percentage values, the actual customer involvement during the releases was ranging from 17.4% to 25.0%, with an average of 20.6%. As it can be seen from figure 2, the actual customer involvement was higher in shorter iterations (two week iterations vs. one week iterations). However, a nearly 100% present on-site customer with an actual involvement rate as low as this is a significant result, since the on-site customer is one of the most controversial topics in extreme programming.

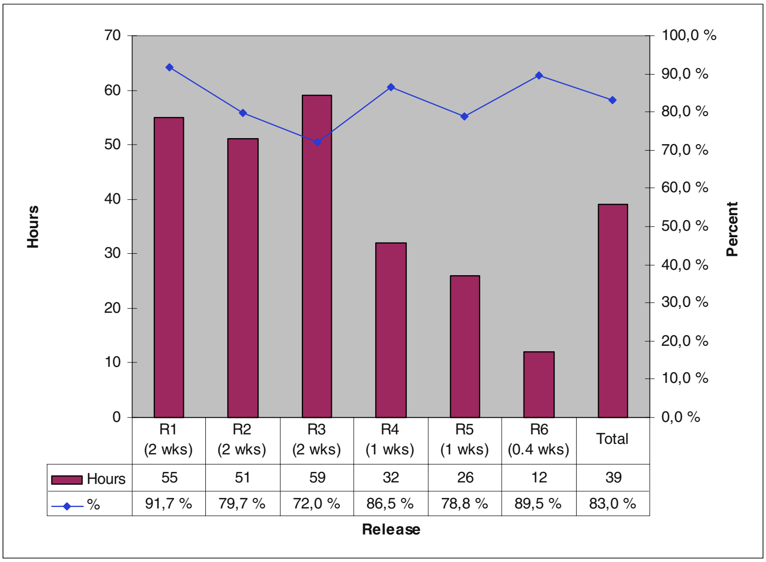

Fig. 1. Customer presence during the project

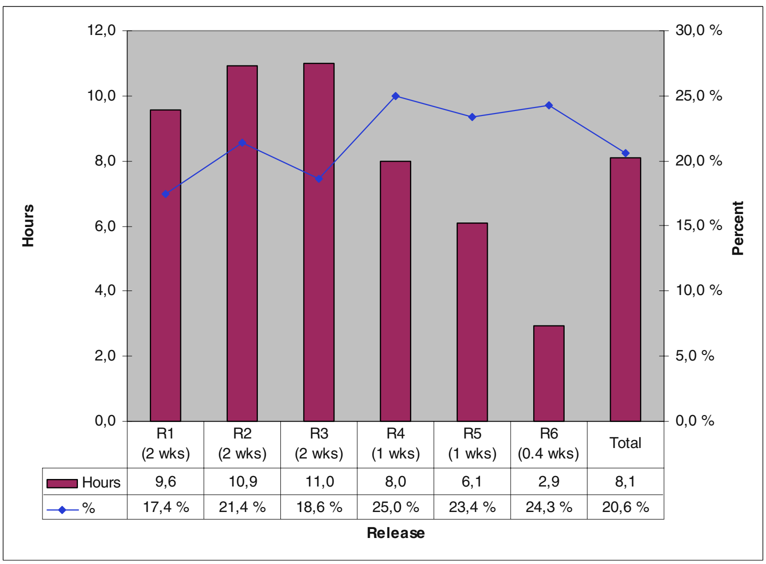

Fig. 2. Actual customer involvement during the project

From the viewpoint of customer effort distribution, participation in planning game and acceptance testing were the two major activities requiring customer's effort (figure 3). Planning game sessions took 42,8% and acceptance testing 29,9% of the total effort. Post mortem sessions [17], which were held at the end of each release cycle, took up 13,4% of the customer effort. The share of project meetings, i.e. planning sessions with the development team during the iterations, and amounted to 8,2% of

the total effort, while 5,7% was required for team coaching activities, due to the customer having the best knowledge of XP and its practices at the beginning of the project. Coaching was needed during the first two iterations, in particular, when developers had something to ask, concerning, e.g., continuous integration or unit testing.

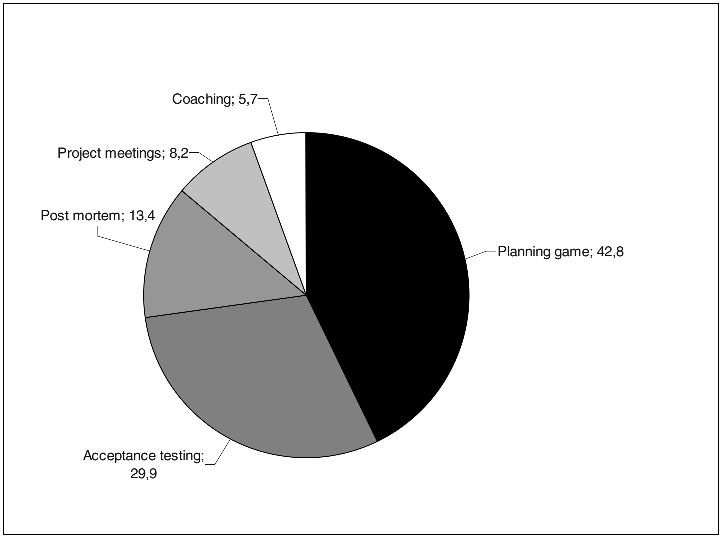

Fig. 3. Customer effort distribution (%)

4.2 Customer Perceptions

The customer did his actual work in the same room with the developers according to the suggestions of XP literature. During the project, the customer made observations of this way of working. It was found out that pair programming generates much more noise than solo programming. In the pair programming mode, the developers read the code out loud and solve problems discussing actively. In solo programming, the developer does not generally share his or her thoughts out loud, thus operating in a more silent mode. During the project, there were usually two pair programming pairs working at the same time and, therefore, the level of noise was always quite high. This was considered to have a disturbing influence on the customer's actual work, especially if the customer was accustomed to work alone in a quiet office space. The second finding was that the developers had quite a low threshold to ask questions when the customer was present on-site. This situation was considered to have both positive and negative effects. According to XP literature, smooth communication between developers and customers is important for project success. On the other hand, if the developers are asking something all the time, the customer may find it hard to concentrate on his actual work. In this kind of development arrangement, the XP customer role should, of course, have the highest priority among the duties of the customer.

In XP literature, the role of XP customer has been argued to be highly demanding. In this particular project, the customer completely agreed with this argument. It was found out that the role of an XP on-site customer requires a strong ability to resolve

issues rapidly. Usually, when developers were asking something, they were also expecting to get their answers straight away. The most common questions and requests were as follows: "What do you think of this implementation?"; "...but the customer decides how we should do this."; and "When you have time, could you test this and that feature?"

4.3 Project Team Perceptions

In the post-project interview, the developers were asked how important they perceived the on-site customer practice. As shown by the following extracts from the interviews, the presence of the customer was highly appreciated:

> *"Very good. If there is some kind of problem, one can just sing it out. There is no need to send any emails or drop in somewhere."*
> *"Got answers fast, if there was something to ask."*

The developers evaluated the on-site customer practice as one of the top 5 positive experiences during the project. For further details on other positive findings, see [19]. It was also looked into whether it was necessary for the customer to be in the same room with the developers. One of the developers answered as follows:

> *"The customer could work, for example, next to the project room. However, it should not take any longer than a couple of minutes to contact the customer."*

Daily participation was also perceived to be highly important. The developers were asked if, e.g., one day a week would suffice:

> *"No, [one day a week] is not enough."*
> *"The customer should be present at least once a day."*
> *"Not necessarily the whole day, but he should be present every day."*

5 Discussion

The results presented in the previous section revealed some important findings regarding the role of an XP on-site customer. XP literature [e.g. 2, 3, 15] generally emphasizes the importance of high level customer involvement during an XP project. However, many authors [e.g. 4, 5-7] also claim that on-site customer involvement is often difficult to realize or even unrealistic due to the required customer work effort. A contrasting result offered by this study was that while the customer was present with the development team at an average of 83%, only 21% of his work effort was required for assisting the development team in the actual development work. The customer effort distribution was quite expected, consisting mainly of planning game sessions and acceptance testing. According to XP literature, these are the main activities of the customer in an XP project.

During the project, the on-site customer found pair programming rather a noisy activity, which may have a disturbing influence on the customer's real work, especially if the customer is accustomed to work alone in a quiet office. This problem could be solved by moving the customer's place of work to a location close to the XP project

room. This solution was also supported by the developers. However, the developers emphasized that it should not take more than a couple of minutes to contact the customer. Moreover, the customer should pay daily visits to the project room. According to Lippert et al.'s experiences [15], this kind of solution is not a problem if there is an agreement that developers can talk to the customer at any time.

Anecdotal evidence suggests that the XP customer role is costly, difficult and demanding. The results of this study support the common belief that the on-site customer's role is demanding and that it requires a strong ability to resolve issues rapidly. Usually, when developers were asking something, they were also expecting to get the answers straight away. The collected data demonstrated that the on-site customer practice offered the team a unique situation to consult him whenever needed. The development team perceived this arrangement as a strong demonstration of the organization's commitment to their work. The project turned out to be a software engineering success. In fact, the team delivered 250% (i.e., 12 user stories identified initially, 34 delivered) more value for the customer organization than originally planned. Furthermore, all this was achieved within the defined delivery schedule. Yet, the developed solution has not been used as actively as intended. The reason for this can be attributed to the relatively poor usability of the system. Yet, all the related stakeholders were happy with the solution when it was under development. The on-site customer also had a lot to say on how the system should function. Consequently, while the experiences were mostly positive, the data also reveals that the on-site customer practice is in danger to create a false sense of confidence towards the system under development. The customer e.g., needs to invest in user-centered design (UCD) activities as the development team may not require them to be conducted.

Table 3. The findings of eXpert case study

Claim, argument or suggestion	**eXpert case study findings**
High degree of customer involvement is required [5, 6, 14]	This study offers contrasting result regarding this argument. While the customer was nearly 100% present with the development team, only 21% of his work effort was required to assist the development team in the development.
The role of on-site customer is very demanding [4, 7]	The study strongly supports this argument. It was found out that the role of XP on-site customer requires a strong ability to resolve issues rapidly. Usually when developers were asking something and they were also expecting to get the answers straight away.
Customer should work in the same room with developers [2-4]	The development team perceived the on-site customer as a strong demonstration of the organization's commitment to their work. However, the customer found pair programming rather a noisy and disturbing activity.

Table 3 presents the main findings of the eXpert case study. Based on these findings, it is suggested that customer could work near to project room, but not necessarily in the same room with the developers. This kind of modified on-site customer arrangement would take both developers' and customer's viewpoints into account; developers could still contact the customer easily and the customer would be able work in a more silent workplace. The customer should also reserve, for example, a one hour per day at the appointed time to discuss face-to-face with the development team.

6 Conclusions

Agile methods have gained a significant amount of attention in the field of software engineering in the last few years. Extreme programming, similarly to other agile software development methods, values close collaboration with customers. The XP on-site customer practice recommends for the customer to be 100% available for the development team. Anecdotal evidence suggests that the XP customer role is costly, difficult and demanding. However, very few empirical studies have been published on the customer role in an XP project. This paper reports the empirical results from a controlled extreme programming case study, in which the customer was present close to 100% of the development time. The results support the common belief that the on-site customer's role is demanding and that it requires a strong ability to resolve issues rapidly. One of the findings indicates that while the customer was nearly 100% present with the development team, only 21% of his work effort was required to assist the development team in the development. However, it is also proven that the on-site customer offers the team a unique situation allowing to consult him whenever needed. The development team perceives this as a strong demonstration of commitment to their work. The data also reveals that the on-site customer is in danger to create a false sense of confidence in the remaining of the customer organization. In addition, the customer found pair programming rather a noisy activity, which may have a disturbing influence on the actual work of the customer, especially if the customer is accustomed to work alone in a quiet office. Based on these findings, it is suggested that the customer should work close to the project room, but not necessarily in the same office space as the developers. This kind of modified on-site customer arrangement would consider both developers' and customer's viewpoints; developers could still contact the customer easily and the customer would be able work in a more peaceful workplace.

There are several threats to the validity of the results. First, the customer was also one of the researchers, which is why his analysis of the results may not be completely objective. However, the customer did not perform the interviews and the developers could thus freely express their views and feelings. Second, the study was based on the analysis of just a single XP case project. However, very few empirical studies have been published on the customer's role in an XP project and, therefore, the data and findings presented in this study can be considered to be valuable for practitioners and researchers in the field. Third, the developers received only a two-day training for the XP method, which can not be seen as sufficient for truly understanding how the method operates. This study is of exploratory nature and aims at gaining a better understanding of the role of the on-site customer for future studies. The findings presented here therefore should be of interest for novice XP teams and customers.

Acknowledgements

The authors would like to thank the eXpert project team for their effort and the three anonymous reviewers for their helpful comments on the early version of the paper.

References

1. Beck, K., *Embracing change with extreme programming.* IEEE Computer, 1999: p. 70-77.
2. Beck, K., *Extreme programming explained: Embrace change*. 2000, Reading, MA.: Addison Wesley Longman, Inc.
3. Jeffries, R., A. Anderson, and C. Hendrickson, *Extreme Programming Installed*. The XP Series. 2001, Upper Saddle River, NJ: Addison-Wesley.
4. Martin, A., J. Noble, and R. Biddle. *Being Jane Malkovich: A Look Into the World of an XP Customer*. in *XP 2003*. 2003. Genoa, Italy.
5. Farell, C., R. Narang, S. Kapitan, and H. Webber. *Towards an Effective Onsite Customer Practice*. in *XP 2002*. 2002. Sardinia, Italy.
6. Griffin, L.A. *A Customer Experience: Implementing XP*. in *XP Universe*. 2001. Raleigh, NC, USA.
7. Stephens, M. and D. Rosenberg, *Extreme Programming Refactored: The Case Against XP*. 2003, USA: Apress.
8. Boehm, B., *Get Ready For The Agile Methods, With Care.* Computer, 2002. **35**(1): p. 64-69.
9. Abrahamsson, P., J. Warsta, M.T. Siponen, and J. Ronkainen. *New directions on agile methods: A comparative analysis*. in *International Conference on Software Engineering (ICSE25)*. 2003. Portland, Oregon.
10. Boehm, B. and R. Turner, *Balancing agility and discipline: A guide for the perplexed*. 2003, Boston, MA: Addison-Wesley.
11. Beck, K. and M. Fowler, *Planning extreme programming*. 2001, New York: Addison-Wesley.
12. Wallace, P., P. Bailey, and N. Ashworth. *Managing XP with Multiple or Remote Customers*. in *XP 2002*. 2002. Sardinia, Italy.
13. Martin, R.C., *Agile Software Development, Principles, Patterns, and Practices*. 2002, Englewood Cliffs, NJ: Prentice Hall.
14. Nawrocki, J.R., B. Walter, and A. Wojciechowski. *Comparison of CMM level 2 and eXtreme programming*. in *7th European Conference on Software Quality*. 2002. Helsinki, Finland: Springer.
15. Lippert, M., S. Roock, and H. Wolf, *Extreme Programming in Action: Practical Experiences from Real World Projects*. 2002: John Wiley & Sons Ltd.
16. Salo, O. and P. Abrahamsson. *Empirical Evaluation of Agile Software Development: A Controlled Case Study Approach*. in *Profes 2004*. 2004. Keihanna-Plaza, Kansai Science City, Japan: Springer.
17. Dingsøyr, T. and G.K. Hanssen. *Extending Agile Methods: Postmortem Reviews as Extended Feedback*. in *4th International Workshop on Learning Software Organizations*. 2002. Chicago, Illinois, USA.
18. Abrahamsson, P. and J. Koskela. *Extreme Programming: A Survey of Empirical Results from a Controlled Case Study*. in *ACM-.IEEE International Symposium on Empirical Software Engineering (ISESE 2004)*. 2004. Redondo Beach CA, USA.
19. Salo, O., K. Kolehmainen, P. Kyllönen, J. Löthman, S. Salmijärvi, and P. Abrahamsson. *Self-Adaptability of Agile Software Processes: A Case Study on Post-Iteration Workshops*. in *XP 2004*. 2004. Garmisch-Partenkirchen, Germany.

Extreme Programming: Reassessing the Requirements Management Process for an Offsite Customer

Mikko Korkala[1] and Pekka Abrahamsson[2]

[1] Department of Information Processing Science
P.O.Box 3000, FIN-90014 University of Oulu, Finland
Mikko.Korkala@oulu.fi
[2] VTT Technical Research Centre of Finland
P.O.Box 1100, FIN-90571, Oulu, Finland
Pekka.Abrahamsson@vtt.fi

Abstract. Software engineering literature and practice have shown that efficient requirements management forms the essence of a successful software project. When the requirements are volatile and the development environment is unstable, Extreme Programming (XP) methodology provides efficient means to cope with requirements through the onsite customer practice. However, for many software companies, having an onsite customer present is difficult to achieve. To solve this problem an Offsite Customer Process Model has been proposed earlier. This paper reports results from a study of a mobile application development project where the model was empirically evaluated. The findings show that the model itself is context-dependent and it has to be adapted to the underlying development process. Based on these findings, an improved model is introduced. It is shown that the new model proved to work efficiently and improved the developer-customer communication mechanisms.

1 Introduction

Research and practice have shown that active customer participation, effective software requirements engineering and requirements management are essential for the success of a software project [1-6]. Extreme Programming (XP) addresses these goals through one of its key practices. On-site customer practice requires the customer to participate actively to the development process, by being always available to answer questions and steering the project to success together with the developers [7-9]. While e.g. the pair programming practise has gained significant interest in the research community [e.g., 10, 11], the onsite customer (or the lack of thereof) has not been under an extensive research, even though the onsite customer is one of the most essential practices of the XP process [8].

Practice has shown that it is sometimes impossible for the customer to participate in the XP project as expected. There can be many reasons for this. Customer can be too valuable to be onsite, or the distance between the customer and the development team is too long [8]. The lack of onsite customer does not automatically lead into a failure, but it does decrease the development pace [8] and increases the project risks, since the planning has to be done further into the future than XP process normally expects [7]. Also in aforementioned situations, the identification and validation of customer's requirements are becoming more important [12].

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 12–22, 2004.

Some solutions for the lack of onsite customer problem have been proposed [see e.g. 8, 13], but these suggestions are not designed to provide a systematic approach to be applied, when the customer is unable to participate to the development as stated by XP. A systematic approach was introduced recently and applied in a small software project with promising results [12]. The aim of this paper is to reassess and improve the Requirements Management Process for an Offsite Customer, based on the results derived from a mobile application development project where the business customer was only partially present.

The results show that the model itself is context-dependent and it has to be applied to the underlying software development process whether it is XP or any other methodology. Based on these results several enhancements and changes to the model are suggested. These enhancements and changes aim to improve the model to be more suitable for a wider context of development processes.

The remaining of the paper is organized as follows. The following section introduces the Requirements Management Process for an Offsite Customer. This is followed by a description of the research design, case study results and a section summarizing the improvements to the model and discussing the future research.

2 A Model for the Requirements Management Process for an Offsite Customer

This section briefly introduces the Requirements Management Process for an Offsite Customer, referred from now on as Offsite Customer Requirement Elicitation Process (OCREP), which was used as the baseline for the empirical study reported in this paper. The motivation for the OCREP is the need for a systematic approach for the offsite customer situation in XP. The following picture provides an overview to the model. Detailed description of the baseline model and its tasks, theoretical underpinnings and the results from the preliminary test can be found in [12].

In *Identify Essential Requirements*- phase, the customer and the developers identify the essential functionalities and concepts by informal interviews. This phase includes also contractual tasks. After this, the requirements are discussed together with the customer and enhanced user stories are evaluated and implemented in the *Evaluation and Implementation of Enhanced User Stories*- phase. Enhanced user stories are detailed descriptions of the requirements of the system, containing also the acceptance tests. This phase of the model serves similar purpose as the Planning Game- practice of XP [7]. The features providing most business value are selected in each iteration by the customer. Since the customer is offsite and thus cannot give instant feedback when it is needed, these features must be discussed carefully. Discussions are supported with user interface illustrations and the acceptance tests are defined together with the customer. Finally, the workload estimates are prepared as in XP [7]. If the customer defines new requirements during the iteration, they are recorded and discussed in the next *Evaluation and Implementation of Enhanced User Stories*- phase.

After the requirements for each iteration are selected, discussed and correctly understood, they can be implemented using the XP practices. During the implementation period, a daily status report is sent to the customer. These status reports aim to provide a brief description of the progress of the project and they contain also descriptions of the implemented features and possible technical problems. If the implemented

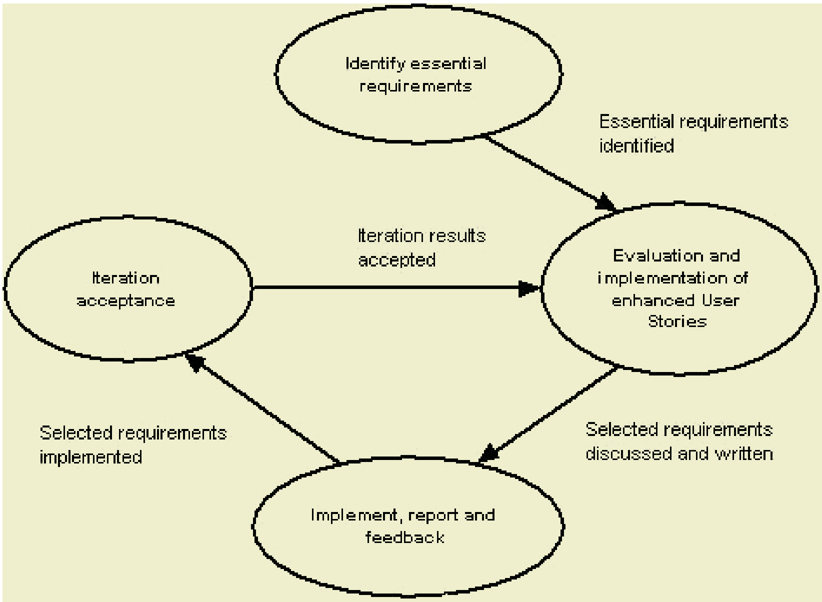

Fig. 1. The Offsite Customer Requirements Elicitation Process [12].

features contain user interfaces, screenshots of those views are attached to the report. The purpose of commenting is to explain the meaning of ambiguous elements to the customer. The customer gives feedback about the implemented features, and thus guides the development and validates whether the features are implemented correctly. At the end of the each iteration the customer personally verifies the implemented requirements, and either accepts the results or proposes necessary changes. The content of the next iteration is also agreed in this phase. If change requests emerge, the workload of the required changes is estimated and the changes are implemented before the implementation of new features. The phases, excluding *Identify Essential Requirements*, are repeated iteratively until the end of the project.

3 Case Study Results

This section presents the principal findings that emerged during the application of the model. The findings related to the different phases of the model are discussed in detail in the following subsections. Originally, the project had one customer, from here on referenced as the original customer. Later two other customers joined the project, forming a customer proxy called a customer group together with the original customer.

The research design was as follows: A team of 5 developers were assigned to implement a stock exchange software for a mobile phone. The code name for the project was zOmbie. The target of the project was to incorporate an access to Helsinki Stock Exchange in to the mobile terminal. The application should enable the user to view, select, browse and sell stocks as well as to display business news. The development

technology was mobile-java. The development team used a development approach specifically developed to support the mobile application development [14] called the MobileD. MobileD contains all of the XP practices as is supplemented with (except for onsite customer). The project lasted 8 weeks in calendar time and the total effort used by the developers was close to 1000 person hours. The reported data is based on empirical results derived from a research journal, interviews and personal on-site observations. This data was used to adjust and evaluate the model shown in the earlier section. The research was conducted as a single case [15] action research study [16]. The model was continuously adjusted and improved throughout the project based on experiences and data.

3.1 Phase 1: Identify Essential Requirements

The development started by agreeing with the communication method and the practice for verifying the results of each iteration. The daily customer communication was done by email and the results were presented at the facilities of the development organization. After the contractual issues were agreed the original customer described the system, dividing its requirements in four major areas of functionality: *see stock prices*, *trend of a chosen stock*, *follow stock prices in real time* and *trading of the stocks*. The team experienced that the informal description of the needed functionality was a working technique for identifying the essential requirements and concepts of the system. It should be noted that only one interview was required to attain the level of adequate accuracy, which supports also earlier findings [12]. For this type of application domain, one interview may therefore be enough to form the “big picture” to get started.

Due to the problems concerning complex features encountered in [12], a crisis communication method task was added in the tasks of the *Identify Essential Requirements*- phase in order to ensure that the problems could be solved together with the customer as soon as possible. Ideally the customer should participate to the problem solving session personally as recommended by Wallace et al. in [13]. If the customer is too remote to participate personally, teleconferencing (or a telephone) could be used as a tool of communication as proposed in [13].

Soon, it turned out that it was impossible for the original customer to participate to the development due to other responsibilities. The team did not receive any feedback from the original customer, thus leaving the group to refine the requirements alone. This had a negative impact on the development team as they interpreted the situation as reluctance from the original customer to fulfil the stated responsibilities. This resulted into a forming of a customer proxy, consisting of two other customers together with the original one. This was organized by the organization’s management since they saw that the team needed the feedback and the original customer was not able to deliver that. The original customer had never participated to an XP project before and had the perception that after having stated the basic requirements, the team should deliver the product within the agreed timeframe. He did not want to have close communication with the development team. The forming of a customer group proved to be a better solution, since at least one customer representative was always able to participate in different phases of the product development. The importance of customer groups is also emphasized in [6]. The establishment of a customer group also increased the developers trust for the success of the project and resulted in better vali-

dation of the requirements. Without a customer group the development decisions would have based on too much guessing. Based on these findings, forming of a customer group was added as a task into *Identify essential requirements*- phase.

As defined in the baseline model, the development team reported their progress on a daily basis. During the project, the customers' side reported that the content of the daily status reports were not understood since they were filled with technical details. Customer group was missing the big picture from the daily reports. Thus, the reports were changed so that the implemented tasks related to a specific user story were identified and reported at a higher level of abstraction. The customers were pleased with this new format of the reports. It is suggested, therefore, that the level of reporting should be decided together with the customer right from the start. The customer can then decide, whether he wants a detailed version of the reports, including the implemented tasks and commented screenshots which are described in [12], or just a general description with or without the screenshots. This division serves only as a guideline, since detailed categories are not yet defined. Based on the findings, the Agree with the feedback level- task is added to this phase. These findings further demonstrate that the baseline model itself is context-dependent. It cannot be presumed that the techniques presented in [12] can be applied as defined, but the level of feedback and a forming of a customer group must be taken into account before the actual implementation can begin. In summary, the improvements for the baseline model are as follows: 1) *Define crisis communication method*, 2) *Form a customer group*, 3) *Agree with the feedback level.*

3.2 Phase 2: Evaluation and Implementation of Enhanced User Stories

As already hinted in [12], the Enhanced User Stories proved to be too rigid, since they were not used in their original purpose. They were originally used as a means of communication, and the developers, based on their ideas of what the functionality of a certain requirement should be. This was based on the presumption that the customer can be too busy to write user stories, or even does not know how to write them as discussed in [17], even though learning is also embraced in XP [8]. Because the Enhanced User Stories were found unworkable, they were discarded. In this research, the discussed requirements were processed into user stories and tasks as defined by Beck in [7].

The procedure in which the selected requirement is discussed together with the customer as stated in [12] was experienced extremely important because of the customer feedback. The team discussed about the prioritised requirements together with the customer, and used graphical support material to clarify the functionality. The team and the customer made illustrations of the user interfaces related to the feature currently under discussion. The user interface illustrations were drawn on 4x6 Post-it notes, each note representing a single view. This technique created lively discussions and exchange of ideas between the developers and the customer, integrating the customer to the development process. If the team did not understand something that the customer wanted, the customer was asked to illustrate the user interface and explain the functionality at the same time. Otherwise the team was responsible for creating the user interface illustrations based on the conversations. These illustrations were discussed together with the customer. These user interface illustrations were proven useful, which is in line with the results concerning the usefulness of the paper proto-

types [e.g. 18, 19]. It was experienced that the user interface illustrations clarified both the user interface and functionality of the software, since the team was able to identify necessary classes and functions that were needed in order to perform the discussed operation. These user interface illustrations improved also reusability and workload estimates. The workload estimates were improved also by the fact that it was possible to identify all the user interface screens required by the feature before the actual implementation. User interface illustrations were also found as an excellent communication method, but it was also mentioned that they did not serve any other meaning than to just illustrate the user interface. Based on these findings, the user interface illustrations can be considered as a useful technique that manages to clarify and validate software requirements. This finding supports the results presented in [12]. After the selected requirements were discussed, they were processed into user stories and further into tasks by the development team.

The essential findings in this phase are: 1) *UI-illustrations are an excellent communication mechanism,* 2) *UI-illustrations provide deeper understanding to the functionality of the system, and* 3) *UI-illustrations improve both reusability and workload estimates and they manage to clarify and validate requirements.*

3.3 Phase 3: Implement, Report and Feedback

While some problems existed (as was shown above), the daily reports provided a very good view of the progress to the customers. This is also in line with the findings presented in [12]. The commented screenshots were not attached to the daily reports because the customer did not want them to be used, despite their usefulness shown in [12].

In spite of this, the team experienced that their usage could have uncovered especially cosmetic errors from the application. Significant amount of deviations found by the customer were cosmetic, e.g. the names of the actions were different from those that the customers expected or wanted. Since the project was completed in time without using the commented screenshots, their usage can be considered unnecessary. While commented screenshots were not applied as suggested, the empirical findings from this project, and the results presented in [12] show that commented screenshots could be used as a requirements validation technique, and thus they remain as a potential means of communication in the model.

The writing of the daily reports took between 5 to 15 minutes per day. The project manager however felt that it was not worthwhile. This was because the reports were written at the end of the day, and it was difficult for the team to decide what to put into those reports. One possible solution to solve this issue is to assign one of the workstation as a repository for reports, and individual developers can then update the reports. The reports can then be sent at the end of the day, but constructed along the day. There is however no results to be shown about this approach, since it was not applied in the project because this issue emerged after the project at the final interview.

Despite the fact that customers found the more general level reports excellent for providing an overview to the progress, which proved that the reports were read, the team did not receive any feedback based on the reports from the customer. Customers perceived that Because the reports only described the implemented features and the possible problems without any specific questions,. Therefore, the customers experi-

enced that they did not had any reason to give feedback about these general descriptions. A simple template for the reports was formed during the project. This template simply consisted of list of tasks related to the user stories that were under implementation, and of miscellaneous issues and possible problems.

The relevant findings in this phase are: 1) *Daily reports are excellent for providing an overview to the projects progress*, 2) *Writing of the reports is not time consuming*, 3) *Screenshots are not essential for the success of the project*, 4) *Do not expect customer feedback without asking specific questions.*

3.4 Phase 4: Iteration Acceptance

At the end of the each release, the customers validated the correctness of the software requirements and functionality by performing tasks that the contents of the iterations were to provide. The validation was performed against the defined acceptance tests, which are identified by the customer and the team in the *Evaluation and Implementation of Enhanced User Stories*- phase of the presented baseline process [12]. Defects and improvement suggestions were written down, and the customer and the developers then agreed on the order of the corrections. It is recommended in [12] that the content of the next iteration is also decided in this phase. However, the requirements to be implemented next were decided after the *Iteration Acceptance*- phase in a separate meeting, based on the underlying development strategy. In addition, the found errors were corrected after the content of the next iteration was accepted. These findings further support the context-dependent point of view to the model, since the application of the model must be adjusted to fit the practices of the organizations software development projects.

The customers' personal participation was experienced mandatory by the team. It was mentioned that the customers' presence at this phase caused pressure for the team to complete the iteration with all the promised requirements. This pressure was experienced as a positive issue. The customers found errors from the software, gave necessary comments and feedback and constructive criticism to the team about the implemented features, thus validating the results of each iteration. It was also stated by a team member that the personal participation, comments and feedback indicated that the customers really cared about the team's efforts and work.

All the possible defects and issues should be recorded properly. In this project, the development team had problems understanding some of the findings, since they were not reasonably well documented, even though they made the notes themselves. In addition, the findings must be categorized as defined by the development organization. It must also be decided together with the customer about the order of the corrections. The improvements to the model are: 1) *Document defects and other issues with care and categorize them*, 2) *Decide when to make corrections*, 3) *Agree with the order of the corrections*, and 4) *Agree when the contents of the next iteration is decided.*

4 The Improved Model and Future Research

The baseline model presented in this paper was improved based on the empirical findings throughout the project. Figure 2 shows the new model. In addition, the customer groups and developers are described. The developers are shown as, filled fig-

ures, and unfilled figures represent the customer. Two figures indicate that they are multiple entities, i.e. many developers and a customer group. Since the Enhanced User Stories were discarded, the name of the *Evaluation and Implementation of Enhanced User Stories*- phase was changed to *Refine Requirements*. *Implement, Report and Feedback*- phase is divided, in order to emphasize that the customer's main role in the phase is just to give feedback from the implemented features and thus to steer the development. Naturally the customer can identify also new requirements during the iteration. These requirements are discussed together in the *Refine Requirements*-phase as defined in [12]. The contents of all phases are summarized in corresponding tables. The practices modified from the original ones are marked with (*), when (!) indicates completely new tasks.

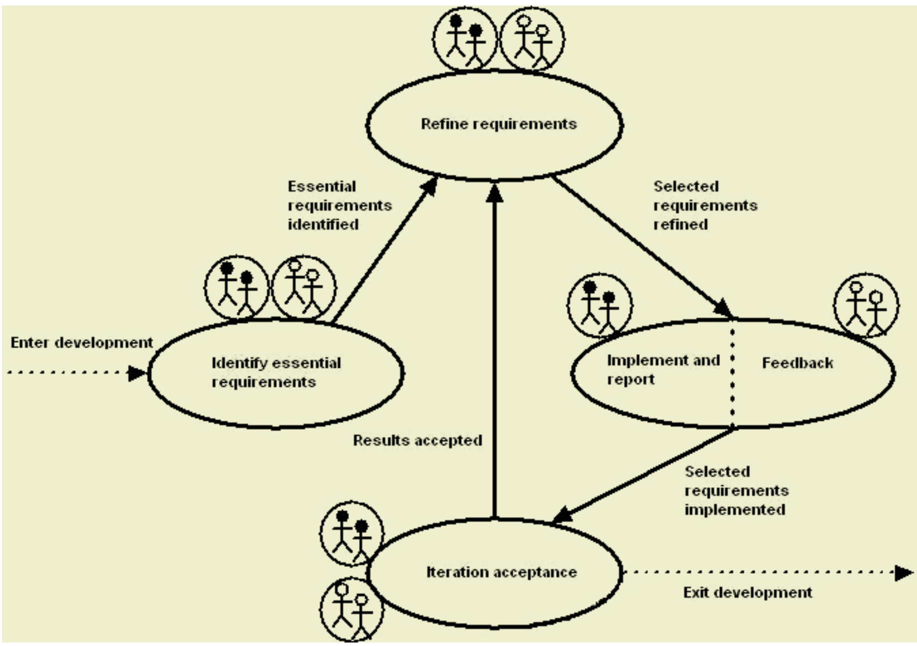

Fig. 2. The improved Offsite Customer Requirements Elicitation Process.

Table 1 summarizes the tasks, actors and outcomes of the Identify essential requirements-phase.

The tasks of the *Refine Requirements*- phase are summarized in the Table 2.

Table 3 summarizes the tasks of the *Implement, Report and Feedback*- phase. Since a crisis communication method is defined, and it's purpose is to establish a communication channel through which the problems are solved together with the customer as soon as possible, the recommendation about refactoring and testing in situations which the development might halt [12], is removed from this version of the model. Task descriptions differ from the previous, but their contents and purpose remain the same.

Table 1. The tasks, actors and outcomes of the *Identify Essential Requirements*- phase.

Task	Actors	Outcome
Agree with the communication method (1st interview)	Customer, developers	Communication method is agreed
Presentation of the iterations results (1st interview)	Customer, developers	Presentation method is agreed.
Describe customer and developer rights (1st interview) **(*)**	Customer, developers	The rights are understood
Define crisis communication method (1st interview) **(!)**	Customer, developers	Crisis communication method is agreed
Establish a customer proxy **(!)**	Customer	Customer group is formed
Agree with the feedback level **(!)**	Customer, developer	Feedback level is agreed
Informal interviews	Customer, developers	The essential requirements are identified
Attain the level of adequate accuracy	Developers	The adequate accuracy level is attained
Define system metaphor	Customer, developers	System metaphor is defined

Table 2. The tasks, actors and outcomes of the *Refine Requirements*- phase.

Task	Actors	Outcome
The customer prioritises the identified requirements	Customer	Prioritised requirements
The evaluation of selected requirements	Customer, developers	Requirements are discussed and understood
Refine selected requirements to user stories and tasks **(*)**	Developers	User stories and tasks
Prepare workload estimates	Developers	Workload estimates

Table 3. The tasks, actors and outcomes of the *Implement, Report and Feedback*- phase.

Task	Actors	Outcome
Implement the user stories	Developers	Implemented user stories
Prepare and send status report to the customer	Developers	Report is prepared and sent
Feedback about the features	Customer	Customer feedback
Make necessary changes based on the customer feedback if needed	Developers	Changed are made
If stories are not understood, ask for immediate support and solve the problems with the customer	Customer, developers	Support request and problem solving

Table 4 summarizes the tasks of the *Iteration Acceptance*- phase.

The model will be further developed and validated in other mobile application development projects in spring and fall 2004.

5 Conclusions

The lack of an onsite customer is a problem for most agile software development projects. This is the case in both large and small development endeavours. While other XP practices have received a great deal of attention, the onsite customer practice

Table 4. The tasks, actors and outcomes of *Iteration Acceptance*- phase.

Task	**Actors**	**Outcome**
Release the implemented features	Customer, developers	The features are accepted or changes are proposed.
Record the possible errors and categorize them **(!)**	Developers	Categorized error log
If change requests emerge, decide when to make changes. **(*)**	Customer, developers	The time for change implementation is decided
Agree with the order of the corrections **(!)**	Customer, developers	Correction order is decided
Decide the contents of the next iteration as in Refine Requirements, or agree when the contents is decided **(*)**	Customer, developers	The content of the next iteration is decided.

has been less studied. An Offsite Customer Process Model [12] has been proposed earlier. This model was applied in a mobile application development project and based on the empirical results, an improved model was suggested. Based on the results of this research and the results presented in [12], the model should be useful to companies using agile methods in their projects. Further evaluation in an industrial environment is however needed.

Acknowledgements

The authors would like to thank the zOmbie- project team for their contribution to this research. Also the customers of the project deserve our gratitude.

References

1. Sagheb-Tehrani, M. and A. Ghazarian, *Software Development Process: Strategies For Handling Business Rules and Requirements.* ACM SIGSOFT Software Engineering Notes, 2002. **27**(2): p. 58-62.
2. van Lamsweerde, A. *Requirements Engineering in the Year 00: A Research Perspective*. in *Proceedings of the 22nd International Conference on Software Engineering. ICSE 2000*. 2000. Limerick, Ireland.
3. Nuseibeh, B. and S. Easterbrook. *Requirements Engineering: A Roadmap*. in *Proceedings of the Conference on the Future of Software Engineering*. 2000. Limerick, Ireland.
4. Vessey, I. and S.A. Conger, *Requirements Specification: Learning Objects, Process, and Data Methodologies.* Communications of the ACM, 1994. **37**(5): p. 102-113.
5. Jain, H., P. Vitharana, and F. Zahedi, *An Assessment Model for Requirements Identification in Component-Based Software Development.* ACM SIGMIS Database, 2003. **34**(4): p. 48-63.
6. van Deursen, A., *Customer Involvement in Extreme Programming*, in *XP2001 Workshop Report*. 2001: Gagliari, Italy. p. 70-73.
7. Beck, K., *Extreme Programming Explained: Embrace change*. 2000, Upper Saddle River, New Jersey: Addison-Wesley.
8. Jeffries, R., A. Anderson, and C. Hendrickson, *Extreme Programming Installed*. 2001, Upper Saddle River, New Jersey, USA: Addison-Wesley.
9. Beck, K., *Embracing Change with Extreme Programming*, in *Computer*. 1999. p. 70-77.

10. Newkirk, J. and R.C. Martin. *Extreme Programming in Practice*. in *Conference on Object - Oriented Programming, Systems, Languages and Applications*. 2000. Minneapolis, Minnesota, United States.
11. Succi, G., M. Stefanovic, and W. Pedrycz. *Quantitative Assessment of Extreme Programming Practices*. in *Canadian Conference on Electrical and Computer Engineering*. 2001. Toronto, Ontario, Canada.
12. Korkala, M., *Extreme Programming: Introducing a Requirements Management Process for an Offsite Customer*. 2004. To be published in Department of Information Processing Science research papers series A, University of Oulu, Finland.
13. Wallace, N., P. Bailey, and N. Ashworth. *Managing XP with Multiple or Remote Customers*. in *3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering*. 2002. Alghero, Sardinia, Italy.
14. Abrahamsson, P., et al., *MobileD: An Agile Approach for Mobile Application Development*. To appear in *OOPSLA 2004* poster session. Vancouver, Canada.
15. Yin, R.K., *Case Study Research Design and Methods*. 2nd ed. 1994: Sage Publications.
16. Avison, D.E., et al., *Action Research.* Communications of the ACM, 1999. **42**(1): p. 94-97.
17. Lippert, M., et al. *XP in Complex Project Settings: Some Extensions*. in *Extreme Programming Conference 2001*. 2001. Villasimius, Cagliari, Italy.
18. Grady, H.M. *Web Site Design: A Case Study in Usability Testing Using Paper Prototypes*. in *IEEE professional communication society international professional communication conference and Proceedings of the 18th annual ACM international conference on Computer documentation: technology & teamwork*. 2000. Cambridge, Massachusetts, USA.
19. Liu, L. and P. Khooshabeb. *Paper or Interactive? A Study of Prototyping Techniques for Ubiquitous Computing Environments*. in *Conference on Human Factors and Computing Systems, CHI'03 extended abstracts on Human Factors in Computer Systems*. 2003. Ft.Lauderdale, Florida, USA.

Global Software Development Project Management – Distance Overcoming

Darja Šmite

Riga Information Technology Institute, Audit and Consulting,
Kuldigas iela 45b, LV-1083, Riga, Latvia
Darja.Smite@riti.lv

Abstract. Global software development becomes one of the most significant trends in the information technology market expansion. This research paper deals with the difficulties and risks examined in global projects in one of the software development companies in Latvia. The research uncovers areas of concern and aims to develop a standardized framework for global project management overcoming the problems brought by distance between the parties involved. The author emphasizes the necessity of unified process framework in global project management and provides her research approach on the way to a deep understanding of global software development.

1 Introduction

In the rapidly changing information era there is always a necessity for new knowledge and resources. Therefore, software development outsourcing becomes a significant challenge of reaching mobility in people and knowledge resources as well as increasing operational efficiency.

Outsourcing as a concept is not a new one. Information system (IS) outsourcing originates from the professional services and facility management services of the 1960s and 1970s in the financial and operation support areas, when computers were very expensive and physically large [12]. Outsourcing nowadays has many different areas (e.g. business process outsourcing, application service provider outsourcing, software development outsourcing, etc.), shapes and models (on-site – on the customer's premises, off-site – on different premises, offshore – in a country with lower taxes), involving two, three or more parties in joint projects.

The author's research field is a narrowing of the versatile topic of outsourcing – software development outsourcing world wide which is well known also as global software development (GSD). Most of the research papers on GSD address the questions of whether to outsource or not ([4], [20], [22]), who, what, why, where and when to outsource, including topics on contractual management ([1], [4], [10]) and risk management ([2], [3], [5]). By this time the question of how to outsource is poorly explored ([14], [21], [18]). Jae-Nam Lee notes that information technology outsourcing has long played an important role in the field, yet outsourcing trends are little understood [11]. Despite its popularity, no research could determine the exact recipe for effective outsourcing performance [14]. According to Jae-Nam Lee, strategic view focuses on competitive advantage, without considering how relationships between an organization and its external environment are managed [11]. In addition most of the researches contribute the general outsourcing service receiver interests

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 23–33, 2004.

whereas the outsourcing providers are competing among each other and accepting the conditions offered by the partner.

Although in many cases global software development can bring the diminishing of the expenses and other benefits as a better knowledge base, extra resources and shared responsibility, entrusting the core strategic values to an external supplier may cause difficulties in the performance, safeguarding of key process confidentiality, a worse control over the project realization and the maximization of the risks of developing dependencies towards an external supplier. That's why one of the major trends in global software development is partnership ([10], [12]). This predicts sharing goals and risks, implementing a unified process management and improving quality of the interactivities.

The background of this research is a survey on global software development process improvement in a software development company in Latvia. The aim of the research is to organize the interdependencies between the company and its partners in other countries into a standardized framework and to improve the software development project performance.

The paper is organized as follows. In the beginning research objectives, background, structure, and approach are described. Then the paper deals with preliminary survey findings. Finally, the author concludes the lesson learned and draws the field for further observation.

2 Research Overview

The research described in this paper is an ongoing project aiming to develop a framework for GSD projects in a Latvian software development company, XYZ. The name of the company and location of its partners are anonymous because of sensitive information of the research.

XYZ is one of the leading software development companies in Latvia with more than 350 employees. The company is oriented towards international market, participating in global projects with partners and customers in Western and Eastern Europe, Baltic and Scandinavia since 90s. Software produced by XYZ is focusing on public sector, telecommunications, insurance and banking, tourism and logistics.

A driving force for this research is a strong need in improving the company's global software projects and achieving effective project performance. The author is involved in ongoing XYZ projects as a specialist responsible for project measurement. Therefore, the research is taking place in a close connection with practitioners. This expands an opportunity of receiving fast feedback from theoretical solution implementation and provides a realistic view on its benefits.

In this chapter the author provides information on the research approach, methodologies used and the structure and aims of the research stages.

2.1 Research Approach

The research approach chosen for the global software development management improvement in XYZ is an action methodology – "learning by doing". Action research methodology aims to contribute both to the practical concerns of people in an immediate problematic situation and to further the goals of social science simultaneously

[17]. In practice action research contains sequential cycles each consisting of four steps: plan, act, observe, and reflect [15].

In this case, the author plans to observe global projects, indicate risks and failures, develop a set of preventive actions and guidelines, test the guidelines in the ongoing projects, measure the results of guidelines implementation, improve the guidelines, etc. (see Fig.1 step Nr.3).

There are several techniques used for data gathering at each stage of the research as observation and case study, measurement through interviews and surveys, documentation analysis.

The expected result of the research is a standardized framework describing global project performance and management. Indicators for measuring research achievements are Client satisfaction; Project completion on time; Project completion on budget; Successful risk management; Process audit appreciation.

2.2 Research Structure

The research can be divided into several stages (see Fig. 1).

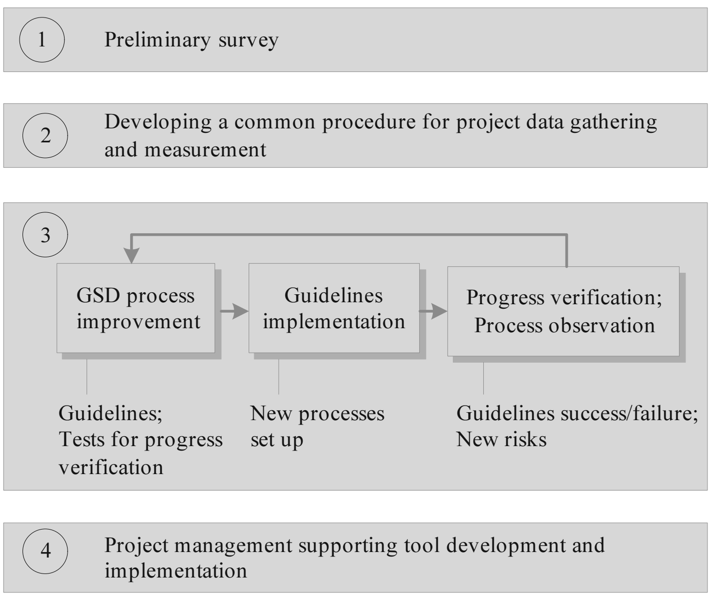

Fig. 1. Research stages

The table below gives an overview on each research stage description, objectives output and current status (see Table 1).

3 The Preliminary Survey Findings

The first stage of the research aims to deepen the understanding of present situation in global project performance in XYZ and consists of several steps as follows.

Table 1. The objectives, output and status of research stages

Research stage	Objectives	Output	Status
Preliminary survey	Aims to deepen the understanding of the present situation in global project management in XYZ	Detailed report on XYZ global projects	In progress
Developing a common procedure for project data gathering and measurement	Aims to define and establish the procedure for project data gathering and success measurement. Prepared using IEEE recommendations [7], ISO requirements [8] and other software engineering best practices ([9], [16], [19]).	Procedure describing data scope, data obtaining sources, responsibility distribution, metrics calculation, reporting sequences.	Developed; submitted for approval.
GSD process improvement cycle	Aims to develop a framework for GSD projects, accumulating best practices and developing guidelines for software development processes and project management.	Guidelines, knowledge database, process descriptions.	Planned
Project management supporting tool development and implementation	Aims to connect all the parties involved in GSD and relieve communication and knowledge share between them.	Supporting tool; data collected by the tool for project measurements, analysis and monitoring.	Planned

1. Formal XYZ documentation (quality management reports, audit reports) exploration in order to point out problems and non-compliances.
2. XYZ questionnaire consisting of a combination of quantitative and qualitative questions, covering all the project stages, indicators, risks and communication liaisons and aiming to understand the present situation in XYZ global projects.
3. Interviews with experienced project managers.
4. Questionnaire for other global software development companies in Latvia.

By this time XYZ documentation exploration and project questionnaire have been conducted. The following chapters provide the findings made during these steps.

3.1 Formal XYZ Documentation Exploration

The author examined documentation containing XYZ software development process and project external and internal audits reports as well as inspection findings from one of the major XYZ external partners in Western Europe.

Most of the internal audit notes concerned a gap in process description (Independent software product testing – 2 notes; Quality system and document management – 6 notes; Software project management – 1 improvement request, 2 notes). In it's turn there was 1 improvement request and 2 audit notes uncovered during the two quality management system external audits according to ISO 9001:2000 standard and TickIT Guide 5 requirements in 2003. All the remarks received during external audits applied to Western Europe projects and concerned gaps in process documentation and quality reviews.

There was a review conducted in 2003 by XYZ major partner management representatives from Western Europe. The review determines areas for project improvement and hence how could the involved parties de-risk the chance of failure. Summarizing the review results, the partners pointed that XYZ teams are working extremely hard, and have significantly improved the capability to complete the work, but are too optimistic in their planning and ultimately are expected to require more time than the current targets allow. A gap in business perspective understanding by testing teams is marked as a serious fault. Project planning is described as extremely high risk and optimistic. As for communication partners noted that the telephone is little used and too much reliance is placed on e-mail that causes delays in turnaround times for solutions. One area of concern is lack of analysis reviews and improper requirement specifications usage that might cause rather serious problems of specific requirement omission. Another remark concerned a gap in problem prioritization during software testing and maintenance stages.

This review was conducted by means of interviewing using standard checklists of issues to cover and ad hoc questioning based on the answers received. In addition, there were breakaway sessions organized to evidence the processes. The author evaluates these findings as one of the most valuable sources of global project exploration at first hand.

3.2 XYZ Questionnaire

The aim of the project questionnaire is to understand the present situation in XYZ global software development projects in order to examine the risks and weak spots of project management. The questionnaire covered the following areas using open-ended and data gathering questions:

- Basic project information;
- Success indicators;
- Methodologies used;
- Responsibility share;
- Development process statement;
- Risks and problems;
- Communication tools used;
- And other.

The author gathered information about 19 global software development projects. By this time the number of projects explored didn't reach the statistical minimum, therefore, the preliminary survey findings cannot be used as comprehensive. Nevertheless, this report serves multiple functions, including summarizing the information collected, facilitating the appropriation of the basic characteristics of the case, pointing the major trends in the company's project management risks and challenges and preparing the input for the further research. The result of this questionnaire is a detailed report on XYZ global software development projects.

The characteristics of the projects can be observed in the table below (see Table 2).

The characteristics of projects being examined allow to conclude that there are several different models of collaboration with the external partners, in particular XYZ being a partner, a contracting party and a direct supplier. In addition, there were pro-

Table 2. Project questionnaire – Observed project characteristics

Characteristics	Values	Number of projects	
Project status	Active	12	63%
	Finished	6	32%
	Stopped	1	5%
Customer location	Western Europe	9	47%
	Central Europe	8	42%
	Scandinavia and Baltic	2	11%
Project type	Software development	13	68%
	Software improvement	4	21%
	Software implementation	1	5%
	Software maintenance	1	5%
Collaboration models	XYZ as a collaboration partner	8	42%
	XYZ as contracting party	6	32%
	XYZ as a direct supplier	4	21%
	Other	1	5%

jects where XYZ was not the only supplier party involved; there were projects involving XYZ developers in on-site development (on the partner's premises); some of the projects were set as time&material, other as fixed price projects. There is a set of detailed observations resulting from the survey in the following chapters. The author deals with only core conclusions because of limited scope of this paper.

3.2.1 Organizational Structure

The first step setting order in project management addressing XYZ global software development projects with the major partner in Western Europe was made by developing organizational structure for software development projects. The structure was presented in a regulative document. According to this document, co-operation process regulation might be implemented by means of organizational structure comprising of the following units:

- Steering Committee (SC), whose main task is strategic and six month project assignment planning, as well as overall co-operation process monitoring;
- Upper Level Change Control Board (UCCB), whose main task is operational planning, Statement of Work co-ordination and project performance monitoring;
- Project Level Change Control Board (PCCB), whose main task is to ensure qualitative and punctual work in compliance with the requirements stated in Statement of Work, as well as change control.

The questionnaire results appeared to be surprising – the regulations are ignored by more than half of projects (see Table 3).

Table 3. Project questionnaire – Organizational structure implementation

Organizational structures	In place	Missing
Project Manager on XYZ side	94%	6%
Project Manager on the other side	68%	32%
SC	32%	68%
UCCB	26%	74%
PCCB	21%	79%

Specifying the results the author would like to note that there was only one project with all organizational structures implemented. As for the Western Europe projects there were no projects with SC, UCCB or PCCB in place. It is surprising because these regulations were developed specially for these projects. In addition, there were XYZ project managers who were not aware about the existence of such structures and their members. The possible problem might concern light-minded attitude of the developer level towards quality management implementation. Wanda J. Orlikowski [18] names "knowing the players in the game" a key factor for successful GSD. In XYZ this appears to be a burden.

The next issue of the questionnaire covered information about XYZ and other side participation in each of the software development processes (systems analysis, design, coding, testing, delivery, and maintenance). The results demonstrated a diverse picture in personnel distribution and participation. This appears to be a field for a deeper examination that will take place during further observation of the global projects by means of interviewing. Author's intention is to find a common way of distributing processes across geographic boundaries in order to perform effectively.

3.2.2 Quality Management

The questionnaire aimed to figure out if global projects explored follow any quality management system. The results of the survey uncovered that most of the projects are held under XYZ Quality Management (37%) or under an adjusted Quality Management (32%). Partner's Quality Management is used in 21% of the projects and only in 10% of the cases each partner uses its own Quality Management.

Further analysis of the survey results revealed that lack of quality management at the partner side appears to be a serious risk.

3.2.3 Communication

Communication is a mediating factor affecting many sides of relationship among remote teams involved in GSD. Erran Carmel [3] considers that given the critical role of effective communication in the successful orchestration of a global software project, it is not surprising that new tactics for addressing distance are being adopted. Therefore, the author aimed to observe communication tools being used in XYZ global projects. The survey results are as follows (see Table 4).

Table 4. Project questionnaire – Communication tools being used

Communication tool	Every day	Often	Seldom	Never
E-mail	58%	42%	0%	0%
Voice mail	5%	5%	5%	84%
On-line discussion groups	0%	10%	21%	68%
Telephone	5%	63%	26%	5%
Audio conferences	0%	10%	10%	79%
Video conferences	0%	0%	0%	100%
Meeting in person	5%	16%	68%	10%

The survey results confirmed the remarks received from XYZ Western European partner concerning a gap in using telephone that causes delays in information turnaround. E-mail appears to be the most common mean of communication (100% of participants uses e-mail either every day or often), while telephone communication follows with only 5% of participants using it every day and 63% of often usage.

The author would like to emphasize that using e-mail as a prior mean of communication between the parties involved in GSD leads to misunderstandings and delays in information turnaround as noted previously. According to Erran Carmel a small issue can take days of back-and-forth discussions over email to resolve, but a brief conversation can quickly clarify the problem before it becomes bigger [3]. The author discussed communication manner problems with experienced global project managers and in many cases they admitted it to be a culture manner problem too. E-mail usage was justified by "totally Baltic" nature – less talking, more doing. In case of global projects, where personal communication is almost absolutely excluded, telephone and electronic means of video and audio conferencing bring humanity in relationship between the distributed teams. Christof Ebert and Philip De Neve summarizing their experience in global software development [5] suggest providing development teams with sufficient communication means, such as videoconferencing or shared workspaces and global software libraries. Nevertheless, the results of the XYZ survey uncovered a seldom application of modern collaboration tools which is often established by the foreign partners.

After the survey the author observed an ongoing European project where videoconferencing is used as the main mean of everyday communication between the remote teams. Both sides involved in the projects evaluate videoconferencing as the best way to manage remote development and progress, and to discuss problems connected with the work being done. Besides it is cheaper then telephone calls and needs few resources for implementation.

Of course, there might be a debate on choosing the best way of communication. Some find sufficient modern electronic means of communication ([3], [5]), other emphasize the importance of face-to-face communication ([18]). Nevertheless, building social networks is generally recognized to be essential in GSD.

3.2.4 Risk Factors

The next step to a deeper understanding of global software projects was achieved by analyzing risk factors and their impact on the project success estimates.

One of the most important factors affecting GSD project success is the quality of systems analysis. The questionnaire results showed that 75% of the participants evaluated the systems analysis results as weak and 10% – as the reason of failure. The quality of systems analysis can be explained by the fact that in many cases the process of requirement gathering is made by on-site team but development by the remote team offshore. Half of the development teams have never met the end-customer during the project development. It claims to a conclusion, that separating systems analysts from the development teams makes the mutual communication and the process of development more complicated, as well as requirement understanding less achievable. A survey on GSD projects conducted by Richard Heeks and his colleagues showed that the most successful projects were those involving members of the remote development team traveling to the customer site for requirements capture [6]. Furthering the research the author will explore problems of systems analysis and requirement transfer between remote teams aiming to develop a common approach to overcome misunderstanding and imperfection.

The questionnaire also aimed to receive evaluation on a set of risks based on extended research papers on risk management in GSD ([2], [3], [5], [18]). The questionnaire provided the following evaluation (see Table 5).

Table 5. Project questionnaire – Risk factor evaluation

Risk factor	Important	Medium Important	Unimportant
Time zones	5%	21%	74%
Low language skills (XYZ)	10%	47%	43%
Low language skills (other side)	0%	5%	95%
Terminology differences	0%	32%	68%
Lack of understanding of tasks assigned	0%	63%	37%
Organization culture differences	16%	37%	47%
National culture differences	5%	32%	63%

Taking in account location of the customer, the relationship with the partners near shore demonstrated relative coherence (risks evaluated as medium important or unimportant). On the contrary the greater is the distance between the partners the more important become the risk factors. Project manager working with partners in a distance of 4’000 km named a set of additional risk factors important in her case, such as lack of quality management, different understanding of cooperation approach, differences in supporting the customer, disagreement on marketing activities.

The most important risk factor according to the questionnaire results is organization culture differences (16% important, 37% medium important) concerning difficulties provided by different organizational structure and responsibility sharing, complicated management levels and other obstacles. Many organizations involved in GSD are not ready to change their internal structure and processes along with new partners. International relationship according to XYZ appears to be an occurrence that needs more attention in contractual management and risk assessment, but not as a driving force for organizational changes, new management approach invention, and process improvement. Therefore, many researches on this field regard partnership and joining values and objectives to be the most beneficial way of managing GSD ([10], [13]).

The factors to pay attention to are also low English language skills of XYZ employees, lack of understanding of tasks assigned, national culture differences and terminology differences.

The author concludes that in global projects where XYZ is involved in GSD as a subcontractor or a partner for a foreign company that coordinates the projects, XYZ is given less responsibility on the project results than so called general entity. Therefore, the attitude of the general entity towards risks and threats that are brought by global appearance may show different results and uncover new areas of concern. Accordingly, the further steps of the survey aim to observe XYZ major partners involved in GSD as general entities.

4 Conclusion and Further Research

This paper deals with a research in progress on global software development. The purpose of this research is to develop a standardized framework for globally distributed project management.

Project management is a great challenge in global software development. There is a plenty of studies on usual software development project management, it seems that everything is well known to any experienced project manager. Unfortunately, there is a number of unexplored and hidden risks that influence project success. To overcome

the problem of distance in GSD, various managers are experimenting and quickly adjusting their tactical approaches [3]. But there is still a gap in a standardized approach and view of global software development process management and problem solution. A common orientation to getting GSD project work done shared across all levels of the company and partners abroad is the key factor for successful performance [18]. It's a great challenge to achieve overall agreement on development methodology across distributed teams as a common ground on which GSD can be implemented for sharing values, goals, and expectations.

The preliminary survey of global projects has shown that distance brings difficulties that are uncommon for on-side development, such as culture, organizational and terminology differences, language skills, communication weakness, and differences in quality management. The main conclusions made by the author are the following:

- there are communication manner problems and undesirability to use new means of interactive collaboration by XYZ employees;
- the distribution of personnel and processes between XYZ and its partners change from project to project, causing problems connected with wrong approach in project process distribution that influence the project results;
- most of responsibility lies on GSD general entity; therefore the achievement of shared goals is complicated;
- informal meetings with project managers have shown that the knowledge and practices used by ones are not being shared across projects and departments, that precludes building and sustaining common ground for GSD.

Deepening the understanding in GSD project management the research will cover the major areas of observation, which as follows (see Fig. 2).

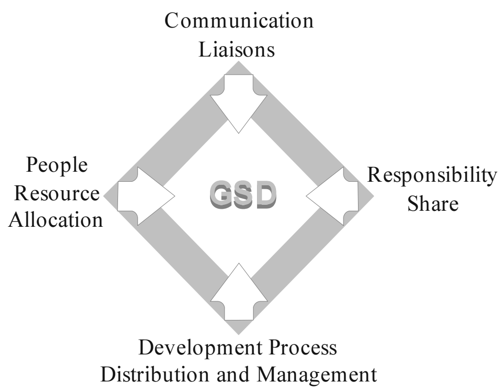

Fig. 2. Areas for developing a common approach

Generalizing conclusions of this research in progress paper the author emphasizes the necessity of a common approach and guidelines for global software development.

Acknowledgement. The author appreciates many valuable discussions with Juris Borzovs and Uldis Sukovskis. Research input was also received from J.Brants, B.Gulbis, V.Karnite, A.Kedrovs, V.Prodnieks J.Rats, U.Reitmanis, I.Safonova, O.Sinica, J.Skrupska, J.Slamovs, E.Taunina, A.Teilans, I.Valdate, M.Vanags, and L.Vascinska.

This research is partly supported by the Latvian Council of Science project Nr. 02.2002 "Latvian Informatics Production Unit Support Program in the Area of Engineering, Computer Networks and Signal Processing".

References

1. Aubert, B., Houde, J.F., Patry, M. and Rivard, S. "Characteristics of IT Outsourcing Contracts. HICSS 2003:269
2. Battin, R.D., Crocker, R., Kreidler, J. "Leveraging Resources in Global Software Development", IEEE Software, March/April, 2001, pp. 70-77
3. Carmel, E., Agarwal, R. "Tactical Approaches for Alleviating Distance in Global Software Development", IEEE Software, March/April, 2001, pp.22-29
4. Department of Information Resources "Outsourcing Strategies. Guidelines for Evaluating Internal and External Resources for major Information technology Projects". Austin, Texas, June 1998.
5. Ebert, C. and De Neve P. "Surviving Global Software Development". IEEE Software, March/April 2001, pp.62-69.
6. Heeks, R., Krishna S., Nicholson B. and Sundeep S. "Synching or Sinking: Global Software Outsourcing Relationships", IEEE Software, March/April, 2001, pp. 54-60
7. IEEE Standards Collection "Software Engineering", 1999 Edition. Institute of Electrical and Electronics Engineers, Inc.
8. ISO/IEC 9126: International Standard ISO/IEC 9126.1991(E) for Software Product Evaluation. Geneva, 1992
9. Jones, C. "Software Measurement Programs and Industry Leadership", Software Technology Support Center, Feb 2001
10. Lacity, M. "Lessons in Global Information Technology Sourcing". Computer, August 2002, pp.26-33
11. Lee, J., Huynh, M.Q., Kwok, C.W. and Pi S.M. "IT Outsourcing Evolution – Past, Present, and Future". Communications of the ACM, May 2003/Vol.46, No.5, pp. 84-89.
12. Lee, J.N.; Huynh, M.Q.; Kwok, C.W. and Pi S.M. "The Evolution of Outsourcing Research: What is the Next Issue?" Proceeding of the 33rd Hawaii International Conference on Systems Sciences, Maui in Hawaii, January 2000, pp.1-10.
13. Lee, J., Kim, Y. "Exploring a Causal Model for the Understanding of Outsourcing Partnership", Proceedings of the 36th Hawaii International Conference on System Sciences, 2003.
14. Loh, L. and Venkatraman, N. "An empirical study of information technology outsourcing: Benefits, risks, and performance implications". Proceedings of the 16th International Conference on Information Systems, Dec. 10-13, 1995, Amsterdam, the Netherlands, pp. 277-288
15. MacIsaac, D. "An Introduction to Action Research," 1995, http://physicsed.buffalostate.edu/danowner/actionrsch.html (14.04.2004)
16. Mochal, T. "End of Project Metrics", TenStep, Inc., 2004
17. O'Brien, R. An Overview of the Methodological Approach of Action Research, 1998, http://www.web.net/~robrien/papers/arfinal.html (14.04.2004)
18. Orlikowski, W.J. "Knowing in Practice: Enacting a Collective Capability in Distributed Organizing", Organization Science, 13:3, 2002, pp. 249-273
19. Pressman, R.S. "Software Engineering. A Practitioners Approach", McGraw-Hill. Inc., 1997
20. Roy, V., Aubert, B. "A Resource-Based Analysis of IT Sourcing", Scientific Series, CIRANO, Montreal, 2000
21. Smite, D. and Borzovs J. "Global Software Development Process Management: Problem Statement", Baltic DB&IS Doctoral Consortium, July, 2004, Latvia
22. Willcocks, L.; Fitzgerald, G. "Guide to Outsourcing Information Technology". Business Intelligence, 1994

Towards Comprehensive Experience-Based Decision Support

Andreas Jedlitschka and Dietmar Pfahl

Fraunhofer Institute Experimental Software Engineering
Sauerwiesen 6, 67661 Kaiserslautern, Germany
{Andreas.Jedlitschka,Dietmar.Pfahl}@iese.fraunhofer.de

Abstract. In today's software development organizations, methods and tools are employed that frequently lack sufficient evidence regarding their suitability, limits, qualities, costs, and associated risks. The need to select the best-suited method, technique or tool in a given business context is becoming more and more important. From a business perspective the trade-off between time-to-market, quality, and cost is a crucial factor for the decision process. While new findings from research await their transfer into industrial practice, systematic assessment, selection and infusion of these findings with regard to business objectives and context is lacking. This paper presents ongoing research towards the development of a decision support system that aims at improving software engineering technology selection by software managers with regard to business goals. The focus of this paper is on presenting the problems at hand, the idea for a comprehensive decision support, and discussing the results of the requirements analysis process.

1 Introduction

It is widely accepted today that software process improvement (SPI), similarly to software development has to be performed in a systematic and managed way. Unfortunately, at least for SPI, this is often not done. For example, it is reported that due to various shortcomings, SPI initiatives often fail. Debou et al. [1] describe three major causes for SPI failure: (1) lack of management commitment, reflected through too few or inappropriate resources, (2) delayed impact upon projects, both for daily practice and performance, and (3) lack of software management skills. Birk [2] reports that SPI initiatives often do not yield the expected results (level of improvement or benefit), because they are (1) performed in an isolated, non-coordinated way, (2) viewed outside their initial context, or (3) not supported by management or individuals.

Within SPI, decisions must be taken regarding Software Engineering (SE) techniques, methods, and tools that have a potential to yield significant improvements. To date, there have been few attempts reported on how to support SPI-managers in their task of decomposing strategic business objectives into activities on a SPI related operational level. The question of which SE technique, method, or tool will fit best into the actual organizational context, is addressed either with support from external consultants, and/or based on the expertise and belief of the manager him-/herself. To find adequate answers, it is, beside other things, necessary to keep track of the state-of-the-art in SE, which is one of the basic "features" of a system in support of SE Decision

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 34–45, 2004.

Support (SE-DS). SE Decision Support Systems (SE-DSS) in general should implement the generation, evaluation, prioritization, and selection of solution alternatives [3].

1.1 Decision Support in Software Engineering

As a consequence, SE-DS is an emerging field [3, 4]. One of the major goals of SE-DS is to support software managers in selecting suitable SE technologies. Suitability implies the existence of a defined level of evidence about the effectiveness of a specific SE technology in a given context. Similar to work done in the area of empirical SE, SE-DS implies data collection, analysis, and modeling. In addition, SE-DS involves model application, possibly supported by software infrastructure.

In the past, research in both empirical SE and SE-DS has focused on evaluating technologies in isolation (e.g., empirical research on inspection techniques [5]). This has produced many results that help software managers to better understand the effectiveness of SE technologies in a stable – but undefined – context. Moreover, much effort has been invested into analyzing results from different empirical studies [6] focusing on the same type of techniques, e.g., [5, 7], hence increasing the level of evidence about the local effectiveness of SE methods, techniques, and tools. This evidence can be used for local SE-DS.

Unfortunately, by relying on nothing but a body of knowledge with – mostly isolated – pieces of local evidence, conclusions about the effectiveness of combinations of SE technologies can hardly be derived without further methodological and tool support. Why is this a problem? It is a problem because

- managers are often not interested in the number of defects that a particular QA technique might potentially find;
- managers are interested in the impact of a particular QA technique on the overall project goals (which include many issues, e.g., functionality, quality, time-to-market, budget constraints, etc.).
- One can even go one step further and say that managers are only interested in the impact of a particular SE technology on the overall business value.

In order to address these broader questions, methods and systems that support comprehensive (instead of local) SE-DS are needed. To give an example, comprehensive SE-DS answers questions of the following type: Which combination of inspection techniques (incl. requirements, architecture, design, code inspections) and test techniques (incl. unit, integration, system, acceptance test) shall be selected in a given business context? In other words, it is not sufficient to compare the effectiveness of different types of code inspections, but the effectiveness of combinations of a specific type of code inspection with other SE techniques along the whole development life cycle [8] must also be taken into account.

One step towards comprehensive SE-DS emerged from the process simulation research community. By emulating the project context with the help of generic process simulators that are instantiated by adding SE technology specific information as needed, many questions similar to the example question above can be answered [9, 10, 11]. There is, however, a practical limitation to this approach. Even though much progress has been made in the last couple of years, it is still very difficult and costly

to come up with company-specific generic process simulators that are sufficiently flexible and valid at the same time.

Therefore, in this paper, we rely on a new approach to comprehensive SE-DS that does not require process simulators but enhances the power of existing SE-DSS that are limited to providing local evidence. Examples of such systems include the ESERNET web-based repository [12].

This paper presents the vision of comprehensive SE-DS, i.e. DS that serves software managers in making context-sensitive SE technology selection decisions aligned to project goals and organizational business objectives. Moreover, as a first step towards realization of a related comprehensive SE-DSS, results of a pilot study on requirements elicitation are presented.

The structure of the paper is as follows. Section 2 briefly summarizes the state-of-art of SE-DS. Then, Section 3 sketches our vision of an improvement framework for comprehensive SE-DS. Section 4 presents both the process and the results of our initial step towards eliciting specific requirements of a comprehensive SE-DSS. The paper concludes with a brief discussion of the results and an outlook to future work.

2 Related Work

We have identified four main areas affecting our work: (1) strategic management, which is most influential for all business-oriented activities, (2) experimentation in SE, (3) decision support and decision support systems in general, for SE, and more specifically for improvement management, and (4) software process improvement and improvement management as application areas of the planed SE-DSS.

2.1 Strategic Management

As mentioned earlier, many improvement activities fail due to a lack of business orientation. Tools from strategic management like the Balanced Score Card (BSC) [13], which is used to break down strategy into operative goals, metrics, and responsibilities through cause-effect-chains, could be used to find interfaces between general improvement driven through Total Quality Management (TQM) [14], the European Foundation for Quality Management - EFQM-model for excellence[1], or “six sigma”[2]. The goal-question-metric (GQM) approach [15] can be seen as one possible candidate for interfacing business excellence with SPI. Eventually, this can lead to improved management commitment. In industrial practice, a combination of management “philosophies” like TQM or EFQM with strategic approaches like BSC as the “steering wheel” and “six sigma” as implementation is proposed (e.g., by Toepfer [16]).

2.2 Experimentation in Software Engineering

The aim of SE experimentation is to provide facts for the suppositions, assumptions, speculations and beliefs that abound in software construction [17, 18]. In other words,

[1] http://www.efqm.org/model_awards/model/excellence_model.htm
[2] http://www.ge.com/sixsigma/

experimentation is the scientific approach to systematically evaluate a SE method, technology, or tool with respect to predefined hypotheses using sound empirical methods. Zelkowitz et al. [19] classify the methods in (1) observational, like project monitoring, case study, and assertion, (2) historical, like literature search, legacy data, lessons learned, and static analysis, and (3) controlled, like replicated experiments, synthetic environment experiments, dynamic analysis and simulation. Other classifications are described by Kitchenham [20], Basili [21], and Travassos [22]. Based on Basili's classification, Travassos proposes the use of in vivo, in vitro, in virtuo, and in silico. In this paper we use the word *experiment* as the generic term for *controlled experiment*, *case study*, *survey*, *pilot project*, and *project*.

2.3 Decision Support

Decisions have to be made every day by everyone. They are made based on the experience, attitude, and intuition of the decision maker. They heavily depend on the context, such as the budget and time available and other environmental restrictions or requirements. Additionally, the information at hand can influence the decision. To give an example: it should be easy for someone visiting a known city with lots of time to decide whether to take the first or the second turn, especially if a map is available, but it would be more complex if one is dropped of in a desert without any water. The complexity of decision-making grows with the impact a decision might have on the decision maker and the environment. This is the case when introducing a new SE technique into the established software development process or building a nuclear power plant.

For an unknown number of years, different research disciplines have looked at decision making in general (e.g., psychologists, system theorists). With the advent (and availability) of computers, more than thirty years ago, the interest in supporting decision making with information technology also arose. Power [23] proposes a matrix with five broad categories of DSS that differ in terms of technology component: communication-driven, data-driven, document-driven, knowledge-driven, and model-driven DSS. Additionally, he distinguishes between user groups, purposes, and enabling technologies.

In SE, many different areas for decision-making can be found along the software life cycle. Ruhe gives an overview on recent research related to SE-DS [3]. He describes five areas in which SE-DS research is progressing, i.e. requirements, architecture and design, adaptive and corrective maintenance, project planning and control, and verification and validation.

For the area of systematic improvement, Birk describes how to use so-called technology experience packages (TEP) in support of technology selection [2]. TEPs are a specific representation of software project experience. Based on a two-step context evaluation with respect to the given situation, appropriate TEPs are ranked and selected by the decision maker. The content of the TEPs can initially be acquired from literature and iteratively be adapted to the specifics of an organization.

Pfahl proposes an integrated approach to simulation-based learning in support of strategic and project management [10]. Following a systems dynamics approach, he shows, that "based on simulations, managers can explore and analyze potential improvements before implementation and empirical evaluation of the related process changes in a pilot project".

Raffo suggests an approach to SE-DS based on Task Element Decomposition and discrete-event simulation [11].

In the area of inspections, Biffl et al. propose a Knowledge Management (KM) framework to support software inspection planning on three different managerial levels, quality management, inspection manager, and inspector [24].

Our initial work towards comprehensive SE-DSS, which we present below, can be characterized as follows. We aim at the development of a data-driven DSS focusing on project planning and control. We are investigating the usage of enhanced TEPs but don't consider simulation of any type (at least in the initial stage). Instead of developing a new framework, or adapting KM frameworks from specialized application domains, such as the one proposed by Biffl et al., the comprehensive SE-DSS will be methodologically backed-up by an improvement framework which is bases in the well-known Quality Improvement Paradigm (QIP) [25].

2.4 Software Process Improvement

In contrast to the mass production of goods, software development projects are more or less unique. Consequently, improvement approaches that are valid in production are not applicable without adaptation to the needs of software developing organizations. Generic frameworks for Software Process Improvement (SPI) are the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM) [26] or more recently the SEI Capability Maturity Model® Integration (CMMI) [27], ISO9000:2000[3], and the Software Process Improvement and Capability Determination (SPICE) [28]. These frameworks are standards for assessing organizational and software process maturity. They can be used for benchmarking against an ideal set of requirements. But they do not propose concrete SE techniques to be used in specific project situations.

The most prominent continuous SPI approaches are SEI's IDEAL Model [29] and the QIP, which can be seen as the software engineering equivalent of TQM. The aim at continuity is reflected through the cyclic nature of the above mentioned improvement approaches.

Although SPI can be considered well investigated, many SPI initiatives fail due to various shortcomings. The most critical one is missing commitment from management and missing integration into the overall strategic improvement of the organization.

3 Comprehensive Decision Support

Comprehensive SE-DS, i.e. selecting most suitable SE techniques along the whole software development project life cycle, requires information about (1) the conditions that have to be fulfilled before a specific SE technique can be applied (precondition) and (2) the project status that is achieved after application of the SE technique. The trade-off as compared to using process simulators is that technology interaction and development context can only be modeled in a black box like manner via pre/post-conditions. Based on the sets of pre-/post-conditions, a graph showing the interplay

[3] http://www.iso.ch/iso/en/iso9000-14000/iso9000/iso9000index.html

between various connectable SE techniques can be drawn. Figure 1 gives an example of the interplay between SE techniques via the pre-/post-condition interfaces.

3.1 A Model for Comprehensive Decision Support

In this example, a company is using an object-oriented software development process, which requires UML diagrams at the end of the design phase. Technology cluster X consists of inspection techniques. Based on results from some empirical studies, a specific inspection technique specifically developed for object-oriented design documents is proposed, e.g., OORT (object-oriented reading technique). Empirical results have shown that OORT finds 75% of the defects of type A+B and 50% of the defects of Type C. Dynamic defects are not found at all.

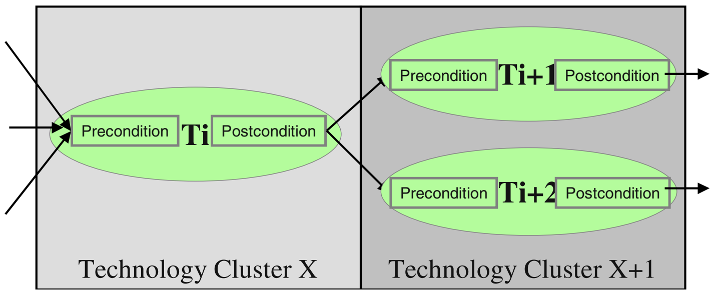

Fig. 1. Interplay between technologies via pre-/post-conditions

The most interesting point is not the effect of inspections as such (local viewpoint), but the impact of inspections on the outcomes of the complete development process (comprehensive viewpoint). In other words, the task is to select that inspection technique which combines best with other QA techniques, i.e., the testing technique.If it is possible to find empirical studies on testing techniques using a similar or the same defect classification, an answer to the selection problem seems to be possible.

Continuing the example of Figure 1, assume that technology cluster X+1 represents testing techniques. Ti+1 be a testing technique that typically finds 75% of type A+B defects and 60% of type C defects, while testing technique Ti+2 typically detects 50% of defect type A+B but 95% of defect type C. Both testing techniques detect 87% of the remaining dynamic defects. In a local decision situation, i.e., if comparing only between testing techniques, one might choose Ti+1. Taking a more comprehensive viewpoint, i.e., considering the combination with the inspection technique Ti, it might be more promising to investigate a combination of Ti and Ti+2, because they produce more synergy with regards to defect detection than the combination of Ti and Ti+1, and also detect more defects than a single SE technique alone.

3.2 Example Schema

To allow the investigation of technology interplay it is necessary to find elements with identical definition in the related experiments. For example, it is only possible to compare certain types of defects between different experiments if the types are the

same. Even in the few existing preliminary guidelines on how to perform [30] and also on how to report experiments [30, 31] this fundamental requirement is not mentioned.

Table 1. Example attributes of a schema for experiments in the area of defect reduction

Element	Attribute	Example
Name of Study	Name	
Object of Study	Object	Inspection process
Purpose	Purpose	Evaluation of sequentially performing code inspections and structural testing
Object (Technique)	T1 .. Ti	Code Reading, Structural Testing
Quality Focus	QF1 .. QFi	(cost-) effectiveness; efficiency; defect coverage
Subjects	Number of	20
	Type of	Students
	Experience	Graduate
Material	Number of	1
	Type of	C Code (262 noncommented lines of code) 13 defects (11 failure, 2 cosmetic)
	Complexity	n.a.
Hypotheses (Ho)	H01 .. H0i	H01: subjects applying code reading within …H02: Teams applying code
Controlled Variables	CV1 .. CVi	Experience with C; inspections; testing
Dependent Variables	DV1 .. DVi	Individual/team defect detection {effectiveness; efficiency},
Defect Type	DT1* .. Dti*	Control flow, cosmetic, initialization, data
Cost		e.g.,Effort spent on object: 1.5 hrs
Results related to the Hypotheses	RH01 ..RH0i	Code reading outperforms structural testing for defect detection {effectiveness, efficiency}
Significance	SH01 .. SH0i	Yes
Validity of the results	Internal	History, Maturation, Selection
	External	Representative subjects, code module
Lessons learned	LL1..LLi	Structural testing may still be considered worthwhile if it focuses on defects of a particular defect class that were missed by inspection
Context	Precondition*	Code available
	Application domain	Statistics
	Project Size	Small
	Project (Process) phase	After Coding
	Postcondition*	X% defects of type A detected Y% defects of type B detected
Publication	Publication	Laitenberger, O.: Studying the Effects of Code Inspection and ST ..
Replication of	Link to parent	n.a.
Aggregation of	Link to children	n.a.

We claim that results from experiments can only contribute to the body of SE knowledge, i.e., the core set of theories about the context-sensitive efficiency and effectiveness of SE technologies, if it is possible to aggregate them with the existing body of knowledge. Therefore, reporting procedures have to follow some minimum requirements, like describing a minimum set of elements and attributes in a standardized way. Experience with a knowlegde management process addressed to solve some of the aforementioned issues is reported, for example, in [12, 32].

In order to come up with an initial proposal for a minimum set of elements that should be reported by experimenters, we surveyed a number (20 out of approx. 500 studies in the area of defect reduction) of experiments on inspection techniques [33]. We found some common elements, which would help to classify the experiments and allow for aggregation of information. Triggered by a similar idea, Vegas et al. [34] describe and evaluate a process for developing characterization schemas to create experience bases, especially suited for testing techniques. By combining our results with those from Vegas et al., we derived a schema for reporting results from experiments in the area of defect reduction. Table 1 gives an overview on such elements with associated attributes. An asterisk indicates attributes that are not commonly available, but would be useful to have; they are considered as part of the initial requirements for future experiments in the defect reduction area, analyzing the combined application of inspection and testing techniques along the SE development life

cycle. To alleviate the filling of the repository the process should be as automatic as possible, e.g., by using text-mining approaches.

For other techniques, e.g., for the design phase, similar schemas have to be developed. Finally, all those schemas have to be merged, or at least an interface between the schemas has to be defined. Also, other kinds of studies like post-mortems or experience reports have to be considered if the whole body of knowledge is to be aggregated [12].

4 Requirements Elicitation for a Comprehensive SE-DSS

Several authors have proposed sets of requirements for SE-DSS on various levels of abstraction. Ruhe, for example, suggests nine categories of SE-DSS requirements [3]. The focus of his analysis is on requirements "that combine the intellectual resources of individuals and organizations with the capabilities of the computer to improve effectiveness, efficiency and transparency of decision-making". In addition, Ruhe defines associated functional components and proposes a generic SE-DSS architecture.

Biffl et al. describe functional and non-functional requirements of a knowledge management system that builds upon a framework to support software inspection planning [24].

All these proposals from literature were a good starting point for our research, but in order to come up with a set of empirically-based requirements specifically aiming at a comprehensive SE-DSS, more work had to be done.

An alternative to Ruhe's architecture, which mainly gives a hierarchy of components related to the requirements (process view), is the standard three-tier architecture, which consists of (1) the user interface layer, (2) the process management layer, and (3) the data management layer.

In this section, we present the results of our requirements elicitation process for a tool in support of SE technology selection [35]. It consisted of (1) expert interviews that were conducted in order to elicit specific requirements for a comprehensive SE-DSS, and (2) deriving additional requirements from lessons learned of a European research project that aimed at establishing an international repository of experience on the effectiveness and efficiency of SE technologies [12, 32, 36, 37]. Based on these two sources of requirements, we order the requirements according to the standard three-tier architecture, and according to Ruhe's generic requirements categories.

The expert interviews were used to elicit requirements of potential end users of a comprehensive SE-DSS, i.e., software managers. Additional requirements that relate to the needs of content (data) contributors, administrators, and the sponsor of the comprehensive SE-DSS were derived from lessons learned we gained with setting-up and running web-based repositories.

Table 2 lists the requirements derived from the pilot study and combines them with the additional requirements. The set of requirements is grouped into five categories: user interface (UI), presentation (PR), content (CO), experience management (EM), and repository (RE). The first two categories correspond to the first layer of the standard three-tier architecture, the third category corresponds to the second layer, and the fourth and fifth category corresponds to the third layer.

Table 2. Requirements for a comprehensive SE-DSS

#	User Interface
UI1	Support for several kinds of graphical / textual presentations
UI2	Low interaction, easy access
UI3	Goal-oriented interaction support
UI4	Alternative interaction modes
	Presentation
PR1	Transparency of decision process (reduction of alternatives, priorities)
PR2	Goal/problem-oriented aggregation of information
PR3	Understandable, self-explaining
PR4	Presentation in diagrams, tables, text
	Content
CO1	Effectiveness/efficiency with respect to quality aspect
CO2	Costs for introduction/applying the technique
CO3	Preconditions that have to be fulfilled prior to the application of the technique
CO4	Context information
CO5	Structured meta information for the content
	Experience Management
EM1	Support for distributed contribution
EM2	Support for distributed quality assurance (distributed content management)
EM3	Support for export of repository data to organizational improvement management systems
EM4	Multi-role management
	Repository
RE1	Cross-linking of experience items
RE2	Case-oriented storing

Table 3. Mapping to Ruhe's idealized requirements

Ruhe's Framework [3]	Specific user requirements for comprehensive SE-DSS
(R1) Knowledge, model and experience management	EM1-EM3, CO1-CO5
(R2) Integration into organization	EM3-EM4, RE1
(R3) Process orientation	CO3-CO5, PR1-PR2, EM4
(R4) Process modeling and simulation	CO3-CO5
(R5) Negotiation	--
(R6) Presentation and explanation	PR1-PR5
(R7) Analysis and decision	PR1-PR2, RE1-RE2
(R8) Intelligence	RE1-RE2
(R9) Group facilities	EM1, EM2

Table 3 shows the mapping of the requirements for the comprehensive SE-DSS to the framework "idealized" requirements (R1-R9) suggested by Ruhe [3]. The instantiation depends on our concrete problem topic, i.e., comprehensive SE technology selection, and usage scenarios, i.e., on-line, individual and strategic decision support for project, quality, and product management. One lesson we learned was that the framework was sufficiently generic to incorporate all of our specific requirements. The following remarks are helpful for interpreting Table 3.

(R1): We assumed that this generic requirement exclusively focuses on the management of the content. We moved requirement related to retrieval, presentation, and explanation – which are often associated with knowledge and experience management – to other generic categories.

(R2): Since we aim at a web-based implementation of the comprehensive SE-DSS, interfaces for export have to be provided to allow for integration into an organization-specific improvement management infrastructure.

(R3): In the envisioned SE-DSS, the decision process will be supported by goal-oriented and problem-oriented interaction facilities.

(R4): Since we do not plan to support simulation, all listed requirements relate to process modeling.

(R5): Since the scenarios offered to the interviewees aimed at individual decision-making, no requirements related to negotiation-support functionality were provided.

(R6): All requirements listed in this category reflect the needs of our target SE-DSS users.

(R7): Since the underlying comprehensive SE-DS was not presented to the interviewees, no specific requirements were given. Nevertheless, we listed those requirements that potentially have an impact on the underlying method.

(R8): Idem.

(R9): The listed requirements stress the distributed character of web-based comprehensive SE-DSS.

The mapping of “our” requirements to both Ruhe’s Framework and the generic three-tier architecture did not yield surprises with regards to their type and novelty.

5 Summary and Future Work

In this paper, we have presented a vision for a comprehensive SE-DSS that supports the selection of appropriate SE techniques by not only looking at a single technique in isolation, but also trying to aggregate information about the impact of the technique on other, related techniques in the software process at hand. It should be mentioned, however, that some of the interviewed subjects were skeptical if they generally would use a DSS, as they felt unable to estimate the actual power of such a system in supporting them in their job. This partly seems to reflect the common fear that information sources potentially create information overload.

There is further research necessary to evolve the vision into a working SE-DSS. So, future work is dedicated to the incremental development of the comprehensive SE-DSS. At each stage, the underlying method and the resulting tool will be evaluated through controlled experiments and surveys among experts from academia and industry. Issues to be evaluated include effectiveness and efficiency of the method and tool support, as well as validity of the delivered information and completeness of the database. At this time a two-step evaluation is foreseen. First, a controlled experiment with students as subjects will be used to get an idea of the efficiency and effectiveness of the approach. A survey among defect reduction experts and consultants should clarify whether, for example, the “base” is representative and whether the approach is able to complement state-of-the-practice with dynamic state-of-the-art information. We plan to report findings from these investigations in the near future.

References

1. Debou, C.; Kuntzmann-Combelles, A.: Linking Software Process Improvement to Business Strategies: Experiences from Industry in Software Process: Improvement and Practice Vol 5, Number 1, March 2000, pp 55-64
2. Birk, A.: A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering; Ph.D. diss., Dept. of Computer Science, University of Kaiserslautern, Germany; Stuttgart: Fraunhofer IRB Verlag; 2000.
3. Ruhe, G.: Software Engineering Decision Support – A new paradigm for Learning Software Organizations. In: Proc. Wkshp. Learning Software Organizations, Springer, 2003.
4. Ruhe, G.: Software Engineering Decision Support: Methodology and Applications. In: Innovations in Decision Support Systems (Ed. by Tonfoni and Jain), International Series on Advanced Intelligence Volume 3, 2003, pp 143-174.
5. Wohlin, C.; Petersson, H.; Aurum, A.: Combining Data from reading Experiments in Software Inspections. In: Lecture Notes on Empirical Software Engineering, World Scientific Publishing Co. Pte. Ltd, Singapore, 2000.
6. Pickard, L.M.; Kitchenham, B.A.; Jones, P.W.: Combining empirical results in software engineering, Information and Software Technology, 40(14): pp. 811-821,1998.
7. Runeson, P.; Thelin, T.: Prospects and Limitations for Cross-Study Analyses – A Study on an Experiment Series. In Jedlitschka, A.; Ciolkowski, M. (eds): “The Future of Empirical Studies in Software Engineering”, Proc. of 2nd Int. Wkshp. on Empirical Software Engineering, WSESE 2003, Roman Castles, Italy, Sept. 2003, Fraunhofer IRB Verlag, 2004. pp. 141-150,
8. Jedlitschka, A.; Pfahl, D. and Bomarius, F.: “A Framework for Comprehensive Experience-based Decision Support for Software Engineering Technology Selection”; In *Proc. of Intern. Conf. SEKE 2004*. Banff, Canada, 2004, pp 342-345
9. Christie, A. M.: Simulation: An Enabling Technology in Software Engineering, CROSSTALK – The Journal of Defense Software Engineering, pp. 2-7, April 1999.
10. Pfahl, D.: An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations, Ph.D. diss., Dept. of Computer Science, University of Kaiserslautern, Germany; Stuttgart: Fraunhofer IRB Verlag; 2001.
11. Raffo, D. M.: Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes on Process Performance, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, UMI Dissertation Services #9622438, 1996.
12. Jedlitschka, A.; Ciolkowski, M.: Towards Evidence in Software Engineering; In Proc. of ACM/IEEE ISESE 2004, Redmondo Beach, California, August 2004, IEEE CS, 2004
13. Kaplan, R. S.; Norton, D. P. *“The Balanced Scorecard. Translating Strategy into Action”*; Harvard Business School Press, Boston, 1996.
14. Arthur, Lowell Jay; *“Improving Software Quality. An Insider's Guide To TQM“*;John Wiley & Sons, New York; 1993.
15. Basili,V.R., Caldiera,G., Rombach,H.D.: Goal Question Metric Paradigm; in: Marciniak JJ (ed.), Encyclopedia of Software Engineering, Vol.1, pp.528–532, John Wiley & Sons, 2001.
16. Toepfer, A. (Ed.): Six Sigma, 2nd ed. Springer Verlag Berlin 2004
17. Juristo, N.; Moreno, A.; Basics of Software Engineering Experimentation; Kluwer Academic Publishers, 2001
18. Wohlin,C.; Runeson,P.; Höst,M.; Ohlsson,M.C.; Regnell,B.; Wesslén,A.; Experimentation in Software Engineering - An Introduction; Kluwer Academic Publishers, 2000
19. Zelkowitz, M.V., & Wallace, D.R.; Experimental Models for Validating Technology. In: Computer May, 1998, pp 23-31.
20. Kitchenham, B.A.; Evaluating software engineering methods and tool; *ACM SIGSOFT Software Engineering Notes*, January 1996, pp. 11-15

21. Basili, V.R.; The Role of Experimentation: Past, Present, Future, (keynote presentation), 19th Internation Conference on Software Engineering, Berlin, Germany, March, 1996
22. Travassos, G.H.; Barros, M.d.O.; Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering; in [JC03] pp. 119-133
23. Power, D.J.: A Brief History of Decision Support Systems. DSSResources.COM, http://DSSResources.COM/history/dsshistory.html, version 2.8, May 31, 2003
24. Biffl, S.; Halling, M.: A Knowledge Management Framework to Support Software Inspection Planning, in [38], pp. 231-249
25. Basili, V.R.; Caldiera, G.; Rombach, H.D.: Experience Factory; in: Marciniak JJ (ed.), Encyclopedia of Software Engineering, Vol. 1, pp. 511-519, John Wiley & Sons, 2001.
26. Jalote, P.; *"CMM in Practice. Processes for Executing Software Projects at Infosys"* Addison-Wesley, Reading; 1999.
27. Chrissis, M. B.; Konrad, M.; Shrum, S.; *"CMMI. Guidelines for Process Integration and Product Improvement"*; Addison-Wesley, Boston; 2003
28. El Emam,K.; Drouin,J.-N.; Melo,W. (Eds.); *"SPICE. The Theory and Practice of Software Process Improvement and Capability Determination"*; IEEE CS, Los Alamitos, 1998.
29. Gremba, J.; Myers, C.: "The IDEALSM Model: A Practical Guide for Improvement" in Software Engineering Institute (SEI) publication, Bridge, issue three, 1997.
30. Kitchenham, B.A.; Pfleeger, S.L.; Pickard, L.M.; Jones, P.W.; Hoaglin, D.C.; El Emam, K.; Rosenberg, J.: "Preliminary guidelines for empirical research in software engineering", IEEE Transactions on Software Engineering, Vol. 28, No. 8 , Aug 2002 pp. 721 -734.
31. Singer, J.: "Using the American Psychological Association (APA) Style Guidelines to Report Experimental Results". In: Proc. of Workshop on Empirical Studies in Software Maintenance, Oxford, England. September 1999. pp. 71-75. NRC 44130.
32. Jedlitschka, A.; Ciolkowski, M.: "Towards a comprehensive summarization of empirical studies in defect reduction"; IESE Report 043.04/E, "Fast Abstract" at ISESE 2004
33. Jedlitschka, A.; Nick, M.: Software Engineering Knowledge Repositories; in [36] pp.55-80
34. Vegas, S.; Juristo, N.; Basili, V.: A Process for Identifying Relevant Information for a Repository: A Case Study for Testing Techniques, in [38], pp. 199-230.
35. Jedlitschka, A.; Pfahl, D.: Requirements of a Tool supporting decision making for SE Technology Selection; In *Proc. of Intern. Conf. SEKE*. Banff, Canada, 2004, pp 513-516.
36. Conradi, R.; Wang, A.I. (Eds.): Empirical Methods and Studies in Software Engineering – Experiences from ESERNET; Springer LNCS 2765, 2003.
37. Jedlitschka, A.; Pfahl, D.: "Experience-Based Model-Driven Improvement Management with Combined Data Sources from Industry and Academia". In: Proc. of ACM/IEEE Intern. Symp. on Empirical Software Engineering ISESE 2003, Roman Castles, Italy, October 2003, IEEE CS, 2003. pp. 154-161
38. Aurum, A.; Jeffery, R.; Wohlin, C.; Handzic, M. (Eds.): Managing Software Engineering Knowledge; Springer-Verlag; Berlin 2003

Discovering the Relation Between Project Factors and Project Success in Post-mortem Evaluations

Joost Schalken[1], Sjaak Brinkkemper[2], and Hans van Vliet[1]

[1] Vrije Universiteit, Department of Computer Science,
De Boelelaan 1083a, 1081 HV Amsterdam, The Netherlands
{jjp.schalken,jc.van.vliet}@few.vu.nl
[2] Utrecht University, Institute of Information and Computing Sciences,
Padualaan 14, 3584 CH Utrecht, The Netherlands
s.brinkkemper@cs.uu.nl

Abstract. Post-mortem project reviews often yield useful lessons learned. These project reviews are mostly recorded in plain text. This makes it difficult to derive useful overall findings from a set of such post-mortem reviews, for example to monitor and guide a software process improvement program. We have developed a five-step method to transform the qualitative, natural language type information present in those reports into quantitative information. This quantitative information can be analyzed statistically and related to other types of quantitative project-specific information. In this paper we discuss the method, and show the results of applying it in the setting of a large industrustrial software process improvement initiative.

1 Introduction

Most software project management methods recommend that projects be evaluated. Although the advice to perform project evaluations is sound, most project management methods offer no guidance as to how to analyze the data collected. This paper fills this gap and describes an exploratory data analysis method to extract useful information from qualitative post-mortem project evaluation reports.

Two kinds of project evaluations can be distinguished [1]: *intermediate project evaluations* that take place periodically during the course of the project, and *post-mortem project evaluations* that take place at the end of a project, when the actual task of the project has been finished.

The objectives of these evaluations differ. Intermediate evaluations are used by senior management to periodically assess whether the project's goals and objectives are still relevant [1] and to monitor project risks [2]. Post-mortem project evaluations on the other hand have the objective *"to evaluate past experience and develop lessons learned for the benefit of future projects"* [1]. The context of this paper is project post-mortem reviews as a means to develop lessons learned for future projects.

The insights gained in the course of the project are made explicit and are recorded in the post-mortem project evaluation report. This evaluation can be useful to the people that have been directly involved with the project, but also to the organization at large. People directly involved in the project may gain an understanding of the factors that attributed to and/or undermined the project's success. One of the strengths of post-mortem project evaluations is that they force people to reflect on their past work and at

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 46–56, 2004.

the same time the evaluations also provide feedback from other team members. These lessons learned can also be used outside the group of people directly involved in the project, to reuse improvements and to avoid pitfalls in future projects throughout the organization.

Before insights from project evaluations can be used by the rest of the organization, they first need to be packaged for reuse [3]. For the organization as a whole post-mortem project reports usually contain too much project specific details. Therefore, the information in post-mortem project reports needs to be consolidated before it can be quickly understood and used throughout the rest of the organization. The consolidation of project evaluations is usually based on software metrics, since software metrics allow for easy consolidation.

Software metrics, both objective and subjective, can be used to evaluate projects. The advantage of objective metrics is that they do not require the judgment of an expert. However, their formal definitions usually require strict adherence to a measurement procedure and frequently require a lot of data to be collected and aggregated in order to measure the attribute. Without the required data no measurement for the attribute is possible, which explains why for many projects not all potentially useful objective metrics are available. Subjective measures on the other hand require no strict adherence to measurement rules, the judgment of an expert suffices. This explains the higher availability of subjective measurements of projects [4].

Even when resorting to subjective measurements for project attributes the distillation of knowledge from past experience is not easy. Without carefull up-front [5])design of the subjective metrics, chances are that the scales of the subjective measurements are meaningless. On top of that potentially interesting project attributes are often missing from the metrics database. This leaves the analyst with missing data and measurements that are meaningless.

To solve the stated problems we propose a new method to explore potentially interesting relations present in project evaluations. Instead of limiting ourselves to just the project metrics database we propose to use the natural language post-mortem project reports as an additional source of data. Using Ishikawa or fishbone diagrams [6] we are able to recode the qualitative information in the post-mortem project reports into quantitative information. This quantitative information can be analyzed statistically to discover correlations between factors. This proposed method has been tested extensively in a case study invovling 55 projects at the internal IT department of a large financial institution.

The remainder of this paper is structured as follows. In Sect. 2, we discuss some related work in this area. We present our method in Sect. 3, and Sect. 4 contains a case study in which we applied the method to a substantial set of real-world post-mortem review reports. We end with our conclusions in Sect. 5.

2 Related Work

Wohlin and Andrews [4] have developed a method to evaluate development projects using subjective metrics about the characteristics of a project, collected using a questionnaire that can even be conducted at the end of the project. They have created a

predictive model for certain success indicators, based on subjective metrics of project factors. Our work differs from their work in that our approach does not even require the collection of subjective measurements at the end of the project. Instead, we extract subjective metrics from qualitative data as found in post-mortem review reports.

As our method places lower demands on the required information, the method proposed in this paper might be applicable to an even wider range of projects than Wohlin and Andrews' method. On the other hand, as our data has less structure, our method results in a large percentage of missing data, which limits the analysis that can be performed on the data (regression model building and principal component analysis are not feasible when our method is used).

Damele et al. have developed a method to investigate the root causes of failures during development [7]. Their method uses questionnaires to obtain quantitative information on failures, which are analyzed using correlation analysis and Ishikawa diagrams. Their method differs from ours in that it uses Ishikawa diagrams to present the results of the analysis and we use the diagrams as an intermediate structuring technique.

In [8] we used Grounded Theory to interpret and analyze data from a large set of semi-structured interviews with practitioners in the area of software architecture. The Grounded Theory method is a qualitative approach to inductively distill theory from a dataset [9]. This approach is not meant to test an existing hypothesis, but provides a method for emerging a theory from collected data. The basic idea of Grounded Theory is to read and reread some textual database, and iteratively 'discover' and label a set of concepts and their interrelationships. In the research described in this paper, we apply a method related to Grounded Theory when populating the fishbone diagrams [6].

3 Method

Before we can describe the analysis method and insights the method attempts to spot, we first need to introduce some definitions. These definitions follow the definitions given by Wohlin and Andrews [4]. A *factor* is a general term for project aspects we would like to study. Factors are either *project factors* or *success factors*. Project factors describe the status, quality, or certain characteristics of a project (e.g. as the testing tool used and team morale), and their value can either be determined prior to starting the project or during the execution of a project. Success factors capture an aspect of the outcome of a project (e.g. the timeliness of a project).

The method described below attempts to expose the effects of project factors on the success factors of a project. The method could try, for example, to discover the effect of using a certain programming language (which is a project factor) on the productivity of those projects (which is a success factor). Our method for discovering insights in natural language post-mortem evaluations consists of the steps listed in Table 1.

3.1 Identify Success Factors

To determine whether a project is a success or a failure, one needs to know what the important aspects (or factors) are of a project in the eyes of the stakeholders. Examples of success factors are timeliness, productivity and stability. Success factors can both be

Table 1. Analysis process to discover relations in post-mortem reports.

Process steps
Identify success factors. Identify the factors that determine the success of a project in the eyes of the stakeholders.
Select project evaluations. Select project evaluations for further analysis. To obtain meaningful results, one might select projects with extreme values along certain success factors.
Identify project factors. Identify repeating patterns in project factors by screening a subset of the selected projects (from step Select project evaluations). These repeating patterns will be structured using an Ishikawa diagram.
Interpret project evaluations. Read and interpret all project evaluations (from step Identify project factors) using the Ishikawa diagram created in the previous step. After the interpretation, the project is evaluated on the project factors that are present in the Ishikawa diagram.
Analyze correlations. Analyze correlations between project factors and success factors. Sort through the correlations between the project factors to find interesting results.

measured objectively and subjectively. The success factors we used in our case study are listed in Table 3.

3.2 Selection of Project Evaluations

Within the scope of an analysis, it might not be feasible to analyze all projects of which a post-mortem project evaluation is available. Therefore we first need to stratify the projects based on success factors identified in the previous step. The stratification process selects a proportion of projects that score high or low on the the success factor and another proportion of projects that score average on the success factor. The stratification selects a disproportionately large group of projects that are an extreme case for one of the selected success factors, as these projects yield most information.

In the stratification process projects with the most extreme values on certain dimensions are found by selecting those projects that deviate more than X standard deviations from the average for that dimension. X is chosen such that the manageable number of projects are selected.

The stratification should not lead to different conclusions than an analysis of a random sample, since stratification does not disrupt the coefficients in a regression equational model as long as the other assumptions of the regression model are not violated [10].

3.3 Identify Project Factors

The open questions in project evaluations do not have numerical answers. We thus have to look for a transformation from the natural language texts available to numerical scores on project factors, which can subsequently be analyzed. We use fishbone diagrams to bring structure in the remarks in the project evaluations to find the underlying

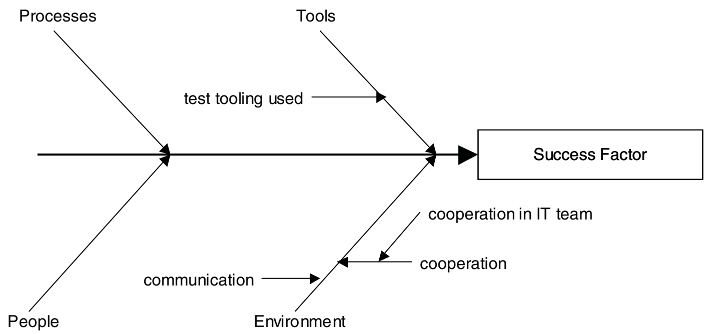

Fig. 1. Example of fishbone diagram containing project characteristics.

project factors in the answers to the open questions. The project factors that are identified as relevant in this step, will be used in the next analysis step to code all selected project evaluations.

Fishbone diagrams, also known as Ishikawa diagrams, are a well-known technique from quality management to detect the causes of disruptions and production problems [6]. To fill the fishbone diagram, we take a sample (say, 30%) from the set of selected project evaluations. From this small set, we select those keywords which best characterize the project factors in the project evaluations. To identify which project factors are relevant we do not need to examine the full set of projects that will be used in the next analysis step, as we expect relevant project factors to occur frequently.

Next, we replace keywords that describe the same pattern by a more general keyword, one which represents all observations from that category. As a rule of thumb for the granularity of keywords we use the following: each keyword must be observed in at least four projects, and at most 50% of the projects must score either positive or negative on this keyword. The reason for the latter restriction is that only keywords which discriminate projects are interesting. We can't learn much from the observation that, say, "user involvement is a crucial factor" if this remark is made for almost all projects.

The keywords found are placed in a fishbone diagram. The main categories we used to organize the diagram in the case study are: processes, tools, people, and environment, see Fig. 1. Our top-level structure was derived from a classification from Balanced Score Card [11] schemes used elsewhere within the company. If no such structure is available up front, it can be obtained as a byproduct of a Grounded Theory-like investigation. The keywords describe the answers to the open questions listed in Table 2. Since these keywords classify the answers to the questions, and not the questions themselves, these keywords do not have a one-to-one relation to the open questions. For reasons of space (we distinguished over 80 categories in the case study), the list of categories is not included in the paper. Figure 1 contains a few typical sub categories (such as: test tooling used) and subsubcategories (such as: cooperation in IT team) we used. Please note that the organization of the fishbone diagram only reflects the classification of the remarks in the project evaluations and does not imply any causal links between project factors and success factors.

3.4 Interpret Project Evaluations

After the keywords and patterns have been distilled from the project evaluations, the subjective interpretation can start. During the subjective interpretation step, selected project evaluations are read by the researcher. This researcher interprets in which categories from the fishbone diagram the remarks from the evaluation fit. Next, it is determined whether the project scores positive or negative on this category. For example, one of the categories we distinguished is change management. A project evaluation report may for instance contain phrases that pertain to the notion of change management. From these phrases, the researcher may deduce that change management has been implemented well for the project, or that problems with respect to change management occurred during the project. Using Likert scaling, we transform natural language text in the answers to open questions into numerical information. For each project, the scores of each category are included in a spreadsheet, together with information from other sources on project aspects. This leaves us with a concise numerical characterization of each project.

3.5 Analyze Correlations

The interpretation of the post-mortem project evaluations yields a spreadsheet with a quantitative encoding of the information from the project evaluation database. As said above, this spreadsheet may be coupled with other numerical information from the project administration database. Using a statistical package, we may next determine correlations between the project evaluations on the one hand, and other quantitative information on the other hand. In our case study, we had quantitative information on productivity, conformance to budget and schedule, and satisfaction of different sets of stakeholders.

The matrix that results from the subjective interpretation of the project evaluations unfortunately has a high number of variables in relation to the number of observations. Normally we would use a technique such as principal component analysis to reduce the number of variables. However in our case a large percentage of the data is missing, which makes principal component analysis infeasible.

Instead of first reducing the number of variables in the data matrix, we directly measure the correlation between the project factors (the independent variables) and the success factors (the dependent variables). This leads to a matrix of correlations between project characteristics and success indicators. We used Kendall's tau [12] measure for the correlation, since this measure is suited for ordinal data. We use pair-wise deletion when encountering missing data, instead of list-wise deletion [10], to make optimal use of the available data.

The correlation coefficients in the matrix are not all based on the same number of observations. Correlation coefficients that are based on a larger number of observations offer more certainty that the observed correlation is really present in the underlying population. This certainty is shown in the level of significance of the correlation coefficient, which can be calculated by statistical packages [13].

Rather than examining the correlation coefficients for all identified project factors, we may opt to restrict ourselves to the most significant coefficients. As the sheer amount

of numbers in the correlation matrix can distract attention from the most influential project factors. The factors can either be selected based on the highest overall level of significance or based on the highest level of significance on a single factor. To calculate the overall significance of a project factor the average is taken of the squared correlation coefficients between a project factor and all success factors.

Note that the statistical significance observed in this type of analysis often is not very high, due to the small sample sizes. As we make multiple comparisons we should apply a Bonferonni or Sidak correction to compensate for the multiple comparisons if we want to use the technique as a conformatory instead of a exploratory technique. As the statistical significance of the results is rather low, we need to have a theory for the correlations observed, in order to derive useful information. Correlation by itself does not imply the direction of the causality.

4 Case Study

In this section we discuss a case study in an organization in which we performed the method described above.

4.1 Context of Case Study

This study has been performed within an internal Information Technology department of a large financial institution. This department employs over 1500 people. The organization primarily builds and maintains large, custom-built, mainframe transaction processing systems, most of which are built in COBOL and TELON (an application-generator for COBOL). Besides these mainframe systems, an extensive variety of other systems are constructed, implemented, and maintained by the organization. These systems are implemented in various programming languages (such as Java and COOL: Gen), run under a variety of operating systems (such as Microsoft Windows and UNIX) and are distributed over different platforms (batch, block-based, GUI-based and browser-based).

4.2 Evaluation Process

The organization has developed its own post-mortem project evaluation method. The evaluation method consists of an online administered questionnaire composed of both open and closed questions. In the evaluation process three groups of stakeholders are addressed: the customer who has commissioned the project, the IT personnel that participated in the project and the involved middle management of the IT department.

At the end of each development project a mandatory evaluation cycle is initiated by the IT project office of the organization. Upon request of the project office the project leader invites involved stakeholders by e-mail to fill out the evaluation questionnaire. When a sufficient number of stakeholders has filled out the questionnaire, the project leader activates an evaluation consolidation routine in the evaluation programme, which anonymizes the responses and calculates averages of all the closed questions.

The evaluation questionnaire contains both open and closed questions. The open questions in the questionnaire are listed in Table 2. As there are over 150 closed questions in the questionnaire, only the categories used to group the closed questions are

Table 2. Case study: Open questions in project evaluation questionnaire.

Question
– What are the 3 most positive points of the project? Explain them. – What are the 3 most important learning points of the project? Explain them. – Can you give 3 suggestions by which the project could have been carried out (even) better? – Was there sufficient input documentation at the beginning of the functional design phase? Which inputs were not available? Indicate the reasons. – Which testing tools were used? What were the advantages and disadvantages? – Was test ware available? If not, what were the reasons? – Which configuration management tools were used? What were the advantages and disadvantages?

included in this paper. The categories of the closed questions are: Time Management, Risk Management, Project results, Project Task (Work), Organization, Work environment, Quality/scope, Project management, and Information, Project. The categories of the closed questions originate from a Balanced Score Card[11] initiative that has been conducted at the organization.

4.3 Bottlenecks

Although the organization has invested large amounts of time in both developing the original evaluation method and evaluating the projects themselves, the organization was unfortunately not able to fully benefit from the lessons learned that lie hidden in the more than 600 evaluation reports that are stored in the project evaluation database.

The organization used the complete information in the project evaluation only as feedback to the project leader responsible for the project. The organization outside the project used the project evaluations only as Balanced Score Card indicators of satisfaction of the commissioning customer and the satisfaction of the IT employees. These overall, consolidated satisfaction ratings made it hard to pinpoint what is going wrong when e.g. the employee satisfaction drops.

The inability to use the project evaluation database can be attributed to four major reasons:

- The database system containing the evaluation database was not available electronically. Manual copying of the data from the database to a statistical tools was required. This made analysis a labour-intensive task.
- As the evaluation method included open questions, some of the answers contained textual answers instead of quantitative data. Textual data is inherently harder to analyze than quantitative data.
- The wording and grouping of the closed questions was inadequate. The grouping of the questions, that was derived from the Balanced Score Card items, was such that many of the categories measured simultaneously different concepts, which makes the interpretation of the average on a category infeasible.

Table 3. Case study: Selection criteria for inclusion of project evaluations.

Selection criterion	*Number of projects selected*
Extreme budget under spending/overspending	10
Extreme planning deviations (finished early and late)	8
Extremely high and low productivity	10
Using COBOL/Telon programming environment	4
Using Java programming environment	4
Using Cool:Gen programming environment	4
Average productivity, no extreme budget or planning deviations	15
Total	55

– As the individual answers on closed questions contribute to the average customer, employee or management satisfaction, one cannot state that the scores on individual questions are independent from the satisfaction measurements. These scale-subscale dependencies make the interpretation of correlation coefficients difficult at least.

The low quality of the closed questions and their clustering combined with the problem of scale-subscale correlations led to the decision to extract project characteristics from the answers to the open questions, using the method outlined in the previous section. Analyzing the open questions had as an added advantage that the answers from every respondent were available, which gave insight into the degree to which the stakeholders in the project agreed on certain issues.

4.4 Experiences

For the analysis of the project information database 55 project evaluations have been selected out of a database containing over 600 project evaluations. The selection of projects included ‘normal projects’, projects with a specific programming environment, and projects that deviated on: productivity, conformance to budget or conformance to schedule. For the deviant projects, an equal ratio of overperforming and underperforming projects has been selected. The exact distribution of the number of projects on the selection criteria is given in Table 3.

4.5 Results

The result of the analysis steps performed in the case study can be found in Table 4. The table contains both the Kendall’s tau correlation coefficients between project factor and success factor, as well as the p-values of those correlation coefficients. The correlation coeffients indicate if there is a strong positive (1), or negative (-1) relation between the factors, or no relation (0). The p-value indicates the strength of the evidence of the relation between the factors, varying form 0 (indicating very strong evidence) to 1 (indicating very weak evidence).

To reduce the correlation matrix we have sorted the factors with respect to the highest overall level of significance. Having sorted the project factors we selected the top 20% from this list.

Table 4. Case study: Results of the correlation analysis on the evaluation matrix.

		Conformance to			*Satisfaction*		
Factor name	*Productivity*	*Budget*	*Schedule*	*Duration*	*Management*	*Employee*	*Customer*
change management	0.05 p=0.71	-0.18 p=0.16	-0.20 p=0.11	0.20 p=0.11	0.33 p=0.006	0.51 p<0.001	0.40 p=0.004
project management	-0.13 p=0.35	0.12 p=0.32	0.39 p< 0.001	-0.26 p=0.02	0.39 p<0.001	0.45 p<0.001	0.10 p=0.42
quality planning	-0.16 p=0.30	0.13 p=0.36	0.34 p=0.01	-0.20 p=0.14	0.43 p=0.001	0.27 p=0.04	0.10 p=0.48
quality schedule	-0.41 p=0.02	0.24 p=0.13	0.23 p=0.13	-0.06 p=0.69	0.10 p=0.52	0.29 p=0.06	-0.19 p=0.30
project control	-0.28 p=0.08	0.17 p=0.25	0.29 p=0.04	-0.34 p=0.02	0.08 p=0.56	0.39 p=0.008	-0.33 p=0.04
testware reused	-0.11 p=0.66	0.50 p=0.02	0.53 p=0.008	-0.38 p=0.06	-0.17 p=0.39	0.39 p=0.05	0.20 p=0.43
quality infrastructure architecture	0.52 p=0.05		0.37 p=0.09	-0.33 p=0.13	0.50 p=0.02	0.08 p=0.70	-0.24 p=0.36
communication efficiency	0.01 p=0.94	-0.36 p=0.06	-0.26 p=0.14	0.30 p=0.10	0.16 p=0.38	0.33 p=0.07	0.25 p=0.20
cooperation	0.08 p=0.58	-0.20 p=0.13	0.25 p=0.06	-0.26 p=0.04	0.22 p=0.08	0.40 p=0.002	0.27 p=0.08
cooperaton within IT	0.39 p=0.03	-0.13 p=0.45	0.46 p=0.006	-0.24 p=0.15	0.59 p<0.001	0.20 p=0.23	0.44 p=0.04
appropriateness team size	0.12 p=0.71	-0.51 p=0.11	-0.93 p=0.004	0.84 p=0.008	-0.14 p=0.65	-1.00 p=0.005	-1.00 p=0.005
team stability	-0.58 p=0.10	0.14 p=0.62	-1.00 p<0.001	0.50 p=0.08	-0.36 p=0.21	0.36 p=0.21	0.18 p=0.56
team stability organisation		0.25 p=0.31	0.36 p=0.13	-0.13 p=0.58	-0.61 p=0.009	-0.27 p=0.24	-0.57 p=0.02
testtool expediter used	-0.27 p=0.03	0.29 p=0.01	0.58 p=<0.001	-0.34 p=0.001	-0.03 p=0.76	0.11 p=0.30	-0.11 p=0.35

The analysis has given us a lot of insight into the quality and organization of the set of closed questions used sofar, and suggested a number of additional closed questions one might ask. This will be used to update and improve the questionnaire, so that, for future evaluations, more quantitative information will be directly available.

At a concrete level, the study showed some interesting, albeit weak, relations between project characteristics and success indicators. For example, high productivity occurs frequently when there is a good cooperation within the team, and when the infrastructure architecture is elaborated well. These relations need further study, though.

5 Conclusions

We have presented a method to analyze qualitative, natural language information as often encountered in post-mortem project reports. The method has five steps: identify success factors, select project evaluations, identify project factors, interpret project evaluations, analyze correlations.

This method provides a structured way to deal with qualitative information such as present in post-mortem project reviews. This information can next be related to other,

quantitative, information present in the company's project database, such as information related to schedule and cost. Information gained from this analysis can be used to improve closed questionnaires that might also be in use, and it gives additional clues that provide useful guidance in a process improvement initiative.

Note that the statistical significance observed in this type of analysis varies, due to the exploratory nature of the analysis. So we need to have a theory for the correlations observed or confirm the results using experiments. Correlation by itself does not imply causality.

Acknowledgments

This research is supported by ABN AMRO Bank N.V. We thank the ABN AMRO Bank for her cooperation. We are especially grateful to Jean Kleijnen, Ton Groen and Cosmo Ombre for their valuable comments and input. We also appreciate the comments of Geurt Jongbloed (Department of Stochastics at the Vrije Universiteit).

References

1. Jurison, J.: Software project management: The manager's view. Communications of AIS **2** (1999)
2. McConnell, S.: Rapid Development: Taming Wild Software Schedules. Microsoft Press, Redmond, WA, USA (1996)
3. Seaman, C.B.: Opt: Organization and process together. In: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research, IBM Press (1993) 314–324
4. Wohlin, C., Andrews, A.A.: Assessing project success using subjective evaluation factors. Software Quality Control **9** (2001) 43–70
5. McIver, J.P., Carmines, E.G.: Unidimensional Scaling. Volume 07-024 of Sage University Paper series on Quantitative Applications in the Social Sciences. Sage Publications, London, UK (1981)
6. Ishikawa, K., ed.: Quality control circles at work. Asian Productivity Organization, Tokio, JP (1984)
7. Damele, G., Bazzana, G., Andreis, F., Arnoldi, S., Pess, E.: Processe improvement through root cause analysis. In Bologna, S., Bucci, G., eds.: Proceedings of the 3rd IFIP International Conference on Achieving Quality in Software (AQuIS'96), IFIP Working Group 5.4 (1996) 35–47
8. van der Raadt, B., Soetendal, J., Perdeck, M., van Vliet, H.: Polyphony in architecture. In: Proceedings of the 26th International Conference on Software Engineering (ICSE2004), IEEE Computer Society Press (2004) 533-542.
9. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Observations. Weidenfeld and Nicolson (1967)
10. Allison, P.D.: Missing Data. Volume 07-136 of Sage University Paper series on Quantitative Applications in the Social Sciences. Sage Publications, Thousand Oaks, CA, US (2002)
11. Kaplan, R.S., Norton, D.: The Balanced Scorecard. The Harvard Business School Press, Boston, MA, USA (1996)
12. Liebetrau, A.M.: Measures of Association. Volume 07-032 of Sage University Paper series on Quantitative Applications in the Social Sciences. Sage Publications, London, UK (1983)
13. Siegel, S.: Nonparametric Statistics for the Behavioral Sciences. McGraw-Hill Series in Psychology. McGraw-Hill Book Company, New York, NY, USA (1956)

Serious Insights Through Fun Software-Projects

Daniel Lübke, Thomas Flohr, and Kurt Schneider

FG Software Engineering, Universität Hannover,
Welfengarten 1, 30167 Hannover, Germany
{Daniel.Luebke,Thomas.Flohr,Kurt.Schneider}@inf.uni-hannover.de

Abstract. Many universities are trying to convey practical experiences to students by conducting software projects. There are many variations of these projects but often the main focus is teaching programming skills and other technical aspects. Most often, the development process is neglected. Because of this, students often experience a "shock of practice" because the industrial daily business depends on many non-technical problems and appropriate skills. These skills have not been taught in university very often and so companies must convey these skills to their new employees.

It would be a better solution if students could experience some or most of these aspects in their education. We think that not all of these can be experienced in the setting of a university. But some of these experiences can easily be made by students. In this paper we present our concepts for conveying these skills within a software project, including techniques like quality gates, walk-throughs and a time-voucher system to simulate the pressure of time in our software project. The project's tasks are a careful mixture of funny and motivating yet very serious aspects. The first phase of the project has already attracted many students' interest.

1 Introduction

In general, software projects being offered by universities cannot give a real impression of the industrial process of software development. There always remain some aspects, which cannot be simulated well in the environment of an university. For example, the missing financial background for such projects is one difference to reality. Most students have other motivations in mind like getting a certificate for the successful participation in the software project. Therefore some way has to be found to motivate students on the one hand and simulate a real development process as good as possible at the same time on the other hand.

In this paper we want to present our ideas and concepts for our software project for students. First we want to give a short impression how the process of software development was taught so far in universities and which problems exist in teaching. In the next section we reflect our own goals and students' goals for the software project. In the fourth section we describe the constraints in which our software project is set in. Besides, we also give a short overview of previous software projects being offered by the University of Hanover. The fifth section is the core section of our paper, because it contains our concepts and ideas for the software project. The next section explains how we want to elicit experiences from students (e.g. for future project set-ups). In

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 57–68, 2004.

the seventh section we give a summarization of similarities and differences between real and simulated software projects. The eight and final section discusses our concepts and contains first reactions of students to these concepts.

2 An Old Problem: Conveying Practical Experience

The problem of teaching good development skills is probably as old as software development itself. The main problem is the difference between the problems developers face in practice during their education: During projects conducted for educational purposes the main problems are technical ones while working on real-life projects, there are various difficulties inherent to communication between different parties and time problems inherent to fast markets.

Although teaching software engineering practices has been subject to lots of research, till today practical courses often focus on teaching basic programming skills, like *how to develop graphical user interfaces* or *connect to databases*. Although universities intend to improve the educational process, the situation in university software development labs has mostly remained the same: Often students get a set of tasks to complete, which may only include programming tasks, like solving one or more specific problems. Better tasks include other complementing processes besides the coding, like for example requirements gathering, documentation and unit testing. The students have a specific time-line until when they have to present their solutions. The instructors will approve the solutions afterwards. In terms of support, the universities are trying to organize as much support for students as possible to help them during their projects because more support is considered to be better education. Because of this, research still continues and new ideas are developed even nowadays [1, 2].

Comparing these set-ups with the constraints and set-ups of real-life projects, one will find many differences, which lead to completely different experiences for the students. First of all and probably most important is that the instructor does not behave like a customer. He spends much more time with the students and normally has not an own inherent interest in the tasks. For example, in real-life projects, the way a problem is solved is not as important as the fact that the customer profits from the project. Furthermore the instructor has more technical skills and often influences the development techniques used for solving the problems. His experiences even lead to a more exact and technical-oriented problem description. In contrast, customers in real-life projects often do not know anything about software design or programming languages. They can only describe what problem they want to have solved and often this description is only sufficiently precise after long discussions. Another difference is the interest of the customer for the project because he has a need for the project which justifies the development time and costs; in university the given programming tasks are normally only fictional tasks which will not be used afterwards. Therefore, the instructors do not have an inherent interest in the solutions.

One possible way to solve these discrepancies is by introducing real-life experiences into the curriculum: Students have to work in industrial projects. While this

scenario allows students to learn real-life problems, it falls short in educational terms. Students can normally only work as programmers and are not allowed to carry out organizational nor management tasks and there is no reflection about used methods and the project cycle afterwards.

Some further possible problems of software engineering teaching in universities have been outlined by Guy Keren [3], which especially include the problem that university staff itself might have too little itself and that the example projects are too small to show the possible effects of software engineering decisions. Because most of our staff has real-life experience and participated in software projects, we want to use our experiences to address these problems.

Beware of these problems some universities also have started to look into possibilities how to improve their teaching. For example, the Georgetown University [4] experiments with setting up a project with many roles. Students get assigned one role and will take part in this project by working in that special field. A completely different approach has been taken by the MIT [5] where software engineering courses focus on web development and related techniques because of the actual importance of that topic.

3 Our Goals, Student Goals

Our software project shall cover different aspects of software development and teachings in university. Especially, we have educational and research goals in mind.

First of all we want to offer students an experience being near to industrial reality, because most students did not take part in industrial software projects before. They never worked in teams and with an economical background in mind. Many students were never confronted with challenging technical software exercises and a time schedule for a development process. We want to give them some insights to these aspects.

Furthermore, we want to offer students interesting software projects which they should develop during the software project. All of our software products contain some interesting design concepts or technical aspects. We want to use all of the finished products for research, teachings and the organization of our department. Therefore, no finished product should be in vain.

We also want to get some research results on how to simulate a software project in university. We plan to realize some concepts to get a software project being nearer to reality always having in mind, that we can not simulate reality in all its facets. At the end of the software project we would like to get some experiences and feedback from students in order to improve our teaching in the next year.

The goals of students differ from our goals. Many students are only interested in getting a certificate for software project. Some other students would like to take a closer look in the process of software development or are interested in technical aspects.

So it is sometimes difficult to match students' interests with our own goals. Therefore, a well-balanced path has to be found how to motivate students and satisfy our goals at the same time.

4 Constraints and Previous Project Set-Ups

The department of computer science of the University of Hanover offers a software project for students being in their fourth semester and studying computer science or a related course of studies. Students taking part in this software project can acquire a certificate and nine ECTS credit points. The certificate only approves the successful participation in the software project and therefore students do not get a mark. The goal of the software project is to improve the students' skills which they got by attending previous lectures: In the first two semesters, the students of computer science at the University of Hanover have two programming classes in which they should learn how to deal with the technical side of programming languages: The first class' subject is the Scheme programming language and in the second class Java is taught. In the third semester the lecture Software Engineering I deals with processes and basic techniques like tests etc.

While students can choose to do one of the classical-style software projects, which are offered by the other department, they can also opt for our approach, which aims to simulate the most important aspects of real-life software projects.

In the summer semester of 2004 approximately 110 students will participate in the software project. Because of our limited resources we can only support 46 students in our department, so we have to cooperate with another department in the university and distribute students between these two departments. In this document we only describe our own concepts.

The entire software project is chronologically limited by the length of the summer semester. Therefore students have only 13 weeks time to complete their task. An extension of this given time is not possible.

During the summer semester students also have to attend different lectures in university. Thus time is very limited for students participating in the software project and attending lectures at the same time.

There are no requirements for the participation in the software project. Any student can take part in it. The organizers cannot reject students, so also students having less knowledge and experience in software development and programming take part in the software project.

Most students are only interested in getting the certificate for the software project being necessary for the completion of their studies. Therefore motivation among some students is low.

There were software projects in previous semesters in the department of computer science for some years. Some projects only concentrated their attention on the solution of difficult technical aspects. Other projects laid their attention on the process of software development.

Also the project management differed from project to project. In the last two years the software project was very restricted. Students of computer science got every two weeks a new exercise to enhance a given software product. Thus creativity and personal contribution to the final software product was quite low.

Some other projects offered students a lot freedom and creativity. Often an inaccurate exercise was given and students had to develop a software product by discover-

ing the mayor demands of the customer who was simulated by the organizers of the software project. In this kind of software projects students were grouped in teams of five to six people. Each group was coached by a teacher giving them some assistance.

5 Our Concepts

The Virtual Company

Our primary set-up is the virtual company "Fungate" specialized in producing games. The students have to work for this virtual company and must solve one of the company's projects. The company's projects will be called tasks in the following to distinguish them from the overall software project. The tasks will be completed by teams of five students each. The students are able to subscribe to one task and therefore build the teams themselves. One student of each team gets elected as the team leader. The team leader is responsible for keeping in contact with the university teachers. The role can be handed over to another team-member during the project, thus allowing all students to learn and experience management roles. Important to note is that building teams really means team-work: The organizational nature of this software projects means that either the whole team passes or fails because the project was finished successfully or not.

The three responsible teachers for the software-project are also participating in this game and therefore are participating in a certain role: For each project there is a customer, a quality manager (QA expert) and a software engineering expert (SE expert). But, as in real life, their time is limited. This is simulated by issuing vouchers which can be used to talk to the experts and to the customer. For example, the students are allowed to talk to the QA manager for 90 minutes, i.e. they have six vouchers with 15 minutes time each. The roles of all participating persons and their relationship can be seen in figure 1.

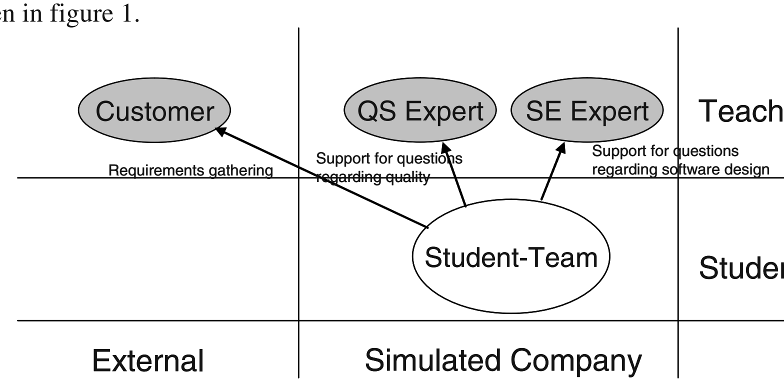

Fig. 1. Roles in our software project

Our project set-up consists of some restrictions which are packaged as requirements of the virtual company. Otherwise, the students are free to solve the tasks as they wish to. Everything which is not explicitly prohibited is allowed.

For example, these restrictions are inherent by templates for the different document types to which the teams have to adhere because they are a company standard. However, they are free to insert any contents into these templates as they wish. Any questions regarding what we expect to be in these documents will be rejected by the teachers; the students will get no specific guidance. This is in clear contrast to a prescribed, fine-grained software process as advocated by Schmedding [6].

The tasks to solve were chosen because we are able to use them for our lectures and they are motivating because of being game-related. The quite unusual and playful set-up was chosen to motivate students to participate. However, the projects we chose are meant serious:

– **GameFrame**, a framework for developing jump'n'run games,
– **NetPong**, a multiplayer and network-enabled game,
– **Risk Manager**, a risk assessment and management tool

Each project has been chosen because we need the product for our own work. For example, the GameFrame framework will be used in lectures and for further projects, the game NetPong has been chosen because it can be used as an example for software which is not really unit-testable and the risk tool has been chosen because we want to introduce risk assessment in further software projects and needed a simple tool for the students. While all these projects can be put in one virtual company without any problems, the projects were carefully selected to offer a broad set of different tasks so that everyone can find a project which he or she likes.

Furthermore, the projects were chosen in such a way that the solution can be efficiently implemented in the Java programming language which the students already know, so that in this practical course the focus can be on social- and process-related problems.

Because we need to support more than forty students with only three teachers, we offered each project three times so that three teams solve the same task. An advantage of this is that even if one group should fail we get the results we already have planned to use in further lectures and projects.

Development Process

Each team starts with a kick-off meeting. During this meeting the project organization will be explained by the customer to the team members. The customer, the QA manager and the SE expert are all attending this meeting. The students' team is allowed to ask any question during this meeting concerning the organization. This meeting is for a long time the only way for the team to speak with all three staff members at the same time. Afterwards, everyone will only be responsible for responding to question concerning his role. For example, if the team has questions regarding quality and the requirements, the team coordinator has to go to two different persons, namely the QA manager and the customer.

To make the situation more realistic, each contact person will pursue different goals: The customer tries to get a solution as fast as possible while the QA manager will insist on a quality solution and the SE manager will demand the best software

design possible. This role-splitting has turned out to be difficult for the teachers because they must concentrate on their simulated interests and knowledge. However, all contact persons will do their best during the time they have to bring the project to a good end.

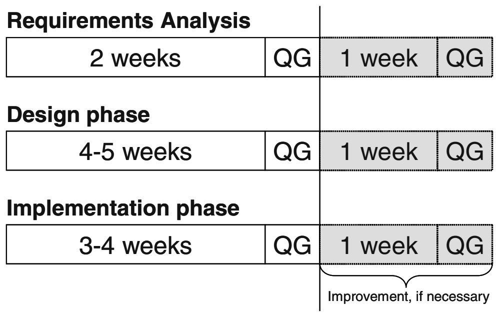

Fig. 2. Project Phases

In order to formalize the process and minimize the projects' and customers' risks, there are fixed quality gates which are dividing the projects into multiple phases which are illustrated in figure 2:

– Analysis phase,
– Design phase, and
– Implementation phase.

After each phase the quality gates as illustrated in figure 3 ensure, that the delivered work-unit for that phase has a minimum quality. If a team does not pass a quality gate it has one week to improve its work and the work will hopefully pass the quality

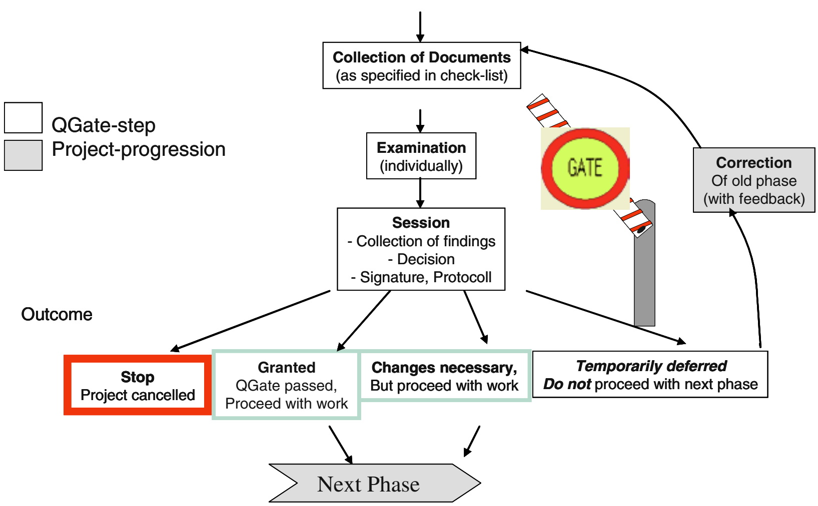

Fig. 3. Process of a quality gate

gate. For each quality gate there is a checklist of items which have to be fulfilled by the delivered work-unit. This list is known by the student-teams and can be used as a quality guideline by them.

In order to improve the students' social skills, especially the ability to criticize and get criticized, the second quality gate implements a walkthrough of the design documents, which requires a team of students to read other team's documents and issue improvement recommendations.

Student Support

By using the voucher system and the strict phases with defined quality gates, stress in terms of time is imposed on the students simulating one of the most inherent problems in software projects today.

However, for offering all students the same opportunities to pass, they can receive unlimited technical support for the development environment Eclipse and other recommended tools by a teaching assistant. We also offer an one-hour introduction to Eclipse.

To help a little bit and to set some standards which the teaching assistant can support, we compiled a CD-ROM containing open-source software which can be used and installed on as many computers as the students wish. This CD-ROM contains the Eclipse development environment including plug-ins for UML-diagrams and a GUI-editor.

6 Reflection and Experience Taken Seriously

The above concepts have been applied to plan and set up our software projects. Most of those concepts were selected in order to make the projects feel more "practical" or more "real" to participating students. In addition to those concepts, we decided to emphasize two aspects *far beyond* (real) industry projects: *reflection-in-action* [7] and conscious *exploitation of experiences* [8].

In a typical software project, most energy and effort are devoted to producing the product. Students may spend several weeks or even months in a company, run into different problems, and still not gain many insights. Reflection on reasons and experiences is often considered overhead.

We try to emphasize reflection and to value experiences: Unlike a typical company, we encourage students to reflect on their experiences. We consider this emphasis on reflection an important *educational value added* on top of "being part of a realistic project". Lewerentz reports on a University projects that exercises even more reflection [9]. Our tight schedule did not allow for such an intense reflexion scheme.

We refer back to Schön´s groundbreaking work on "breakdowns" that cause "reflection in action" [7]. Roughly speaking, practitioners of demanding disciplines (like software engineering) need to be stopped in the middle of their work and forced to reflect when they are still mentally involved. The closer reflection is tied to working (action), the more authentic and useful its results.

Cockburn [10]compares software projects with playing a game and argues each project must both (1) win its actual game and (2) prepare for the next. Reflection is one approach to do the latter, and to stay successful in the long run.

Reflection-in-action is a concept to ***elicit*** experiences. Our previous work in systematic learning from experience [11, 12] has taught us lessons and mechanisms on ***what to do*** with insights and experiences once they have surfaced.

The primary technique we will use to activate [13] and elicit experiences is LIDs [14]. LIDs was developed at DaimlerChrysler Research and Technology and has been applied there. It is a light-weight approach to support a group of peers in capturing and documenting their (common) experiences gained in a common task. The group is guided through a sequence of topics. Participants are asked to talk about their activities and their experiences in chronological order. Associated documents that are mentioned are linked and, thus, contextualized. The entire approach is highly optimized to be easy, fast, and in line with the cognitive needs of all stakeholders. All steps and details of LIDs are described in [14].

We emphasize reflection and we support it through LIDs. We want (1) students to think more about reasons; (2) we want them to *practice reflection* and see LIDs as a simple way to support it; and (3) we want to learn from students´ experiences in order to further improve our project set-up.

In future research we want to interpret the LIDs within the FLOW project, analysing experience, knowledge and project flows and study how to improve the project setup by modifying them.

7 Real, Simulated, and Dissimilar Project Attributes

We claim to make our student projects more "realistic" than many typical University projects. However, we want to provide more learning opportunities than a typical industry project. We have selected a small set of aspects to make the projects feel "real": some of them just happen to be similar to real projects anyway. Other aspects would be dissimilar at a University, but we decided to simulate reality through simple mechanisms (like vouchers). Many other aspects were considered too difficult, too costly, or not important enough to reward simulation. They were accepted to stay dissimilar.

Table 1 lists a number of crucial aspects. For each, we indicate how similar our projects will be to real Industry software projects. Some are similar by themselves; others are made somewhat similar through simulation; and a third group is not even attempted to be made similar. We provide rationale for our choice.

8 Discussion and Conclusions

Software projects are a good and widely-used possibility to convey practical elements of software development to students. Because of this they are included in most curriculums. We developed a new concept to conduct these projects and are using it in our own teaching: by using some focused activities and simulations we want to con-

Table 1. Handling project aspects that determine how "real" a project feels

Aspect	Similarity	Remark/Reason
Motivation	Dissimilar	Money/Promotion cannot be offered at a University. Students could choose the project "they liked" (intrinsic motivation). Nine Credit Points towards their degree was their main extrinsic motivation.
Work Environment	Dissimilar	Most students work at their Laptops and from home. A co-located office environment could not be offered.
Reflection in student projs.	Dissimilar	Educational purpose – would be beneficial to Industry projects, too!
Customer is not Supervisor	Simulated	Through (artificial) separation of roles that are often merged in University. Very important for realistic interactions in requirements engineering.
Limited Customer Contact	Simulated	University staff tends to be "always available". Customers are not. It is easy to simulate a real situation, and it makes a big difference.
Internal Quality Organization	Simulated	Companies have quality departments. We installed a person (an advisor) for each project to represent "internal quality assurance and management"
Customer needs Product	Simulated	When University generate "synthetic project tasks" or just adopt and repeat "real" projects, genuine interest of customers (often: supervisors) is lost. We accepted only projects that the "customer" needed for his work.
Quality Gates	Similar	Many projects and their progress must be monitored by a few quality people. Comparable criteria call for general mechanism, such as quality gates.
Strict Deadline	Similar	Simply do not use potential mechanisms to relax the deadlines (a frequent temptation at Universities)
Responsibilities	Similar	Teams receive some templates, a few deadlines and standards. There is a project leader. Teams are free to organize as they wish. They are responsible for result.
Heterogeneous Teams	Similar	Student teams, as well as professional software projects, showed very different skill profiles.

vey as many interesting and useful experiences as possible with maintainable effort. Simulation in this concept is important to bridge the gap between the conditions in reality and those which are predominant in universities.

Our students have accepted our concept and are eager to profit from the aspects relevant to practice. Their first feedback and our first impressions were positive:

- Students positively mentioned that the simulation aspect is interesting and will prepare them better for their work. Also, they assume that the supervision and support they will receive will be better than with standard set-up projects,
- Students attended very well prepared to each costumer meeting making the meeting very straight-forward and productive. This included question lists and even the usage of voice recorders to save a transcript of the meeting.

However, the vouchers had some unforeseen implications so far. Because the tasks are not completely documented and the teams have to gather requirements, they perceive time as very limited and are using vouchers with too much care. This leads to very short and hectic meetings and teams interrupting the customers during their presentation of ideas.

Especially interesting for us was to see, if our game projects really motivated the students. Our expectation was that developing a game is more interesting for students; however, this aspect was mentioned only rarely as a motivating one. One student even opted for the risk management tool because she does not like games at all.

While planning the role system we assumed that this would be an easy element of our simulation, but in practice it has been very difficult for us: The customer role implicitly requires that the customer does not know anything about technical aspects and must therefore redirect such questions to the QA/SE-expert. But we do have the knowledge in these areas and must be very concentrated to deny any requests for aspects not included in the specific role. Students do not have such problems here, although they always try to ask their questions to any person because they are used to do so and normally can ask everyone.

Although the project has not finished yet, the initial feedback was very positive and we think our approach will prove to be of value. But we must never forget that we have to reflect after each conduction and fine-tune the simulation. This is especially true for the voucher system where we have to change it in a way that hectic and time-awareness is reduced to a normal level.

From our point of view team-work with groups of five students and the diversification of roles is especially useful. As an orientation for students and for us to see, if the teams are still on target, the concept of a schedule with phases and quality gates seems to be supporting and offer guarantees.

Important for the supporting teachers is the new climate of support they give the students. At the beginning there were lots of short but very demanding meetings requiring a high level of concentration. However, in the long run we expect that the time dedicated for student support is the same as in classical software projects.

For companies it would be really interesting to analyse if time spent with customers is so important and not available or if other restrictions are more dominant. Accordingly we could adjust these restrictions and simulate it with our voucher system. The reflection within companies could easily reveal more or other critical aspects of projects. Especially simply the knowledge of resource-restrictions can improve efficiency; the limitations imposed by us led to a high awareness and careful usage. The same holds up for other resources like time in companies.

All in all, we would recommend our project set-up for conveying project experiences to students. It certainly has to be adapted to the environment where it is conducted but the general impression is a very positive one.

Our simulation furthermore revealed that students appreciate courage for more playful set-ups, take it seriously and consequently profit from it. For further improvement of our set-up we welcome any comments and suggestions!

References

1. Deepti Suri, M.J.S. Incorporating Software Process in an Undergraduate Software Engineering Curriculum: Challenges and Rewards. in 17th Conference on Software Engineering Education and Training (CSEE&T 2004). 2004. Norfolk, Virginia (USA).
2. James C. McKim, H.J.C.E. Using a Multiple Term Project to Teach Object Oriented Programming and Design. in 17th Conference on Software Engineering Education and Training (CSEE&T 2004). 2004.
3. Keren, G., Why Do Universities Fail Teaching Software Engineering?, 2002, http://users.actcom.co.il/~choo/lupg/essays/software-engineering-and-uni.html.
4. Blake, M.B. and T. Cornett. Teaching an Object-Oriented Software Development Lifecycle in Undergraduate Software Engineering Education. in Sofware Engineering Education and Training (CSEET2002). 2002. Northern Kentucky, Ohio, February 2002: IEEE Computer Society Press.
5. Abelson, H., Teaching Software Engineering, 2001, http://philip.greenspun.com/teaching/teaching-software-engineering.
6. Schmedding, D. Ein Prozessmodell für das Software-Praktikum. in Software Engineering im Unterricht der Hochschulen (SEUH 2001). 2001. Zürich: dpunkt.verlag.
7. Schön, D.A., The Reflective Practitioner: How Professionals Think in Action. 1983, New York: Basic Books.
8. Schneider, K.B., Victor R.; von Hunnius, Jan, Experience in Implementing a Learning Software Organization. IEEE Software, 2002. **19**(3).
9. Lewerentz, C. and H. Rust. Die Rolle der Reflexion in Softwarepraktika. in Software Engineering im Unterricht der Hochschulen (SEUH 2001). 2001. Zürich: dpunkt.verlag.
10. Cockburn, A., Agile Software Development. 2002: Addison Wesley.
11. Houdek, F. and K. Schneider, Software Experience Center. The Evolution of the Experience Factory Concept., in International NASA-SEL Workshop. 1999.
12. Schneider, K., What to Expect from Software Experience Exploitation. Journal of Universal Computer Science (J.UCS). www.jucs.org, 2002. **8**(6): p. 44-54.
13. Schneider, K. Active Probes: Synergy in Experience-Based Process Improvement. in Product Focused Software Process Improvement (PROFES 2000). 2000. Oulo, Finland: Springer.
14. Schneider, K. LIDs: A Light-Weight Approach to Experience Elicitation and Reuse. in Product Focused Software Process Improvement (PROFES 2000). 2000. Oulo, Finland: Springer.

Software Process Improvement in Small and Medium Sized Software Enterprises in Eastern Finland: A State-of-the-Practice Study

Ilmari Saastamoinen and Markku Tukiainen

Department of Computer Science , University of Joensuu
P.O.Box 111, FIN-80101 Joensuu, Finland
{Ilmari.Saastamoinen,Markku.Tukiainen}@cs.joensuu.fi
http://www.cs.joensuu.fi/

Abstract. Software Process Improvement (SPI) has been proven to increase product and service quality as organizations apply it to achieve their business objectives. Improvement needs of small organizations are same as larger organizations, i.e. they want to achieve better results in software projects, product quality, and customer satisfaction and put an end to the project overruns and failures. However the resources of the small company for SPI-work are often limited and external support seems to be essential. Companies are lacking experience and knowledge how to define and implement appropriate improvement plans and actions. The paper presents current results of software process assessment and improvement work done at University of Joensuu in cooperation with small and medium-sized software companies.

1 Introduction

Software process assessment and improvement is widely acknowledged as one of the most important means for achieving competitive and effective software industry. Different organizations have published and supported state of the art approaches like ISO 15504 [12], CMMI [4] and Bootstrap [18]. One of the goals of these models is to support transition of best practices into software industry. However, among the small and medium-sized organizations the awareness of these models has been weak and even if a small company knows the models and recognizes the improvement needs, their resources - both financial and personnel related - are often limited. Typically small companies are also lacking experience and knowledge on how to define and implement appropriate improvement plans and actions. They encounter difficulties applying these models to their software process improvement (SPI) work because models reflects the software practices of large software organizations [2]. Thus appropriate tailoring of these models is needed [2].

During the years many SPI models for small businesses have already been developed (see, for example [1,2,10,16,20,23]) but the implementation of software process improvement itself is often difficult. Many studies have reported success factors and implementation issues for SPI (e.g. staff and management commitment, business orientation, measurement) in large organizations (see, for example, [6, 21, 22, 26]) and also in small organizations (see, for example [6]).

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 69–78, 2004.

A number of studies have also investigated the state of the practice of software development work and impacts of software process improvement. A good general overview of the state of the practice of software engineering, mainly in large software businesses, can be found on IEEE Software (November/December 2003), special issue for software engineering state of the practice [11]. Also many large organizations have reported their experiences of SPI (see, for example, [5, 7, 8, 17]) and a number of studies have reported experiences of SPI work in small companies (see, for example, [3, 9, 15]).

This article presents the process and results of an empirical study of software process assessments in tSoft-project [25]. The goal of the tSoft-project is to improve the productivity and competitiveness of small and medium-sized software companies in Eastern Finland by assisting them to improve their software engineering processes and working practices. tSoft offers SPI-consultation, education and technology watch services for the participating companies. The focal areas of consultation are software process assessments, assistance in improvement planning and implementation of new practices and processes. Other project's major areas, technology watch and education, are targeted to support process improvement and implementation by updating personnel skills and knowledge on software engineering methods and technologies. The project is managed by the department of computer science at University of Joensuu. Industrial partners are small and medium-sized software companies from Northern Carelia region in Eastern Finland. At the moment the project is at the end of its first phase and participating companies are implementing their planned improvement actions according to their software process improvement plans.

The rest of this paper is as follows: In Chapter 2 we will describe the profiles of the participating companies and in Chapter 3 we will present the assessment process used. Then in Chapter 4, we will present assessments' main findings and study the software process strengths and improvement opportunities for the participating companies. Finally in Chapter 5 we will draw some conclusion and present future directions for the tSoft-project.

2 Profiles of the Participating Companies

Altogether eight software enterprises participated in the assessments during the year 2003. Following background information is collected by using questionnaire forms which company representatives independently filled and returned to the authors.

The companies represent different sizes, ages and application domains of software industry. Six of the companies were small independent software houses and one was a unit of larger software organization. The eighth participant was a software department of an industrial manufacturer. The total number of software work-related employees in participating companies was 92. The smallest software personnel was 3 and the largest company had 35 employees in software. The average number of software workers was 11.5. In 2002 the total revenue of the organizations was circa EUR 9 million excluding the industrial manufacturer. The total average revenue per company was circa 1.5 million EUR.

In the year 2002 companies had typically 5 software projects and the average team size of the projects was 3-5 employees. Most of the organizations and projects had one physical location. The total percentages of production activities of organizations were as follows: new development 40%, maintenance 32 % and other 28 %.

Types of software systems and applications the organizations were developing varied. Most companies (6) developed application software for others, but five of the organizations had also distinct software products of their own and/or they developed commercial software for open markets. The more detailed breakdown (%) is as follows: Company representatives were able to select as many application types as adequate for company development work.

- Application software for other companies (75 %)
- Commercial software for open market (63 %)
- Application software for internal customers (50%)
- Application software for own purposes (38 %)
- Embedded software for own system products (38 %)
- Embedded software for customer's system products (25 %)
- Subcontractor (13%)
- Other (13 %)

Typically the companies' key customers operated in the sectors like education, municipal services and construction industry, but companies had also markets from state authorities, agriculture and food products, military and customer services (retail) sectors. The typical number of production versions of companies' software was one (in four companies) but some of the companies had also more production versions (three companies had 2-5 versions). The most typical development model used was rapid prototyping and incremental or evolutionary model, but also classic waterfall model existed widely. Most used tool of the tool usage in software development were related to documentation (all of the companies used) and project management (in 7 out of 8 companies). Only one of the companies used tools for testing.

3 Assessment Process

In order to study current state of processes and practices in the participating companies, we conducted software process assessments. Process assessment examines the processes used by an organization to determine whether they are effective in achieving their goals [13]. The results of assessments may be used either to drive process improvement activities or process capability determination. This is done by analyzing the results in the context of the organization's business needs and identifying strengths, weaknesses and risks inherent in the processes.

In our case the main goal of the assessments was to find out improvement needs of the organizations and support their process improvement work. The assessment process contained five basic activities: planning, data collection, data validation, process rating, and reporting. A set of templates were also prepared for assessment process including an assessment plan, kick-off presentation, assessment report, presentation of the assessment results and feedback form. Assessment forms used during

on-site assessment sessions were acquired from Finnish Software Measurement Association [24]. In the planning phase of the assessments, survey-forms were also used to study improvement needs and priorities of the companies. Assessments consisted two different parts. In the first part the organizations made self-assessments using KYKY model, and in the second part SPICE-assessments were conducted by trained assessors (authors) in co-operation with software professionals of the companies. The assessments were run during spring 2003.

The *KYKY* model is an overview of organizational or project level quality and process management practices. The model is developed by STTF [19] using ISO9001, SPICE, CMM 1.1 and various other sources. It covers seven process areas and contains 46 questions. Each question were assessed by company representant using a SPICE scale of N (not achieved, <15 %), P (partially achieved, 15 %-50 %), L (largely achieved, 51%-85%), and F (fully achieved, >85 %). The process areas are:

- Process oriented operation (8 questions)
- Customer-supplier processes (6 questions)
- Software engineering processes (6 questions)
- Project management processes (7 questions)
- Support processes (6 questions)
- Organizational processes (5 questions)
- Process improvement (8 questions)

We used the model as an overview study of software company's quality and process management practices at an organizational level. We asked the representative of an organization to fill in the KYKY form. Then we analyzed the answers and wrote a report which included also a comparison with the other participating companies. The KYKY assessments were done before the SPICE-assessments and it helped also to set constraints for the SPICE assessments.

SPICE (ISO 15504) [12] is an international standard for process assessment. We used SPICE-conformant assessment method from Finnish Software Measurement Association in our assessments. The method used is based on the technical report version of the SPICE standard [13,14] published in 1998. The SPICE-assessments produced both an analysis of current capability level and improvement opportunities. A total of 24 people from the companies participated in the SPICE-assessments during the spring 2003. Main data collection methods during assessments were document reviews, interviews and discussions with companies' software professsionals. Because we didn't want to consume too much of the company time, we set out to perform assessment in one day, we had to restrict the assessment scope. We selected five key processes as follows: project management (MAN.2), software requirements analysis (ENG.1.2), configuration management (SUP.2), quality management (MAN.3), and subcontractor management. As a rough guidance for process selection we used level 2 processes from staged CMMI model [4] for software because it provides a good roadmap for improvement. Another constraint for assessment was the capability level of the processes. We performed questions and ratings relating only to levels 1 and 2 although ISO-15504 provides rating levels from 0 (incomplete) to 5 (optimizing).

4 Assessment Findings

In this section the main findings from the assessments, both KYKY and on-site SPICE assessments, are reported but first we present the results from the survey which was done before assessments. These results describe the strengths, weaknesses, improvement priorities and risks of the organizations in their own opinion and experience.

Before the assessments company representatives were asked to fill a questionnaire which inquired which three processes or activities were most important strengths and weaknesses of the company. Figure 1 presents the most frequently mentioned issues of the survey. The competency of company personnel was a critical strength of the companies. Project management activities were also considered as well as technical know-how of the personnel. Three most important weaknesses were in the areas of testing, product management and project management. The improvement opportunities the companies were interested in were related to testing, requirements management and customer cooperation. The three main risks were unsatisfied customers because of defects, schedule and cost overruns and possible changes in personnel.

Strengths	**Weaknesses**
– Know-how of the personnel (experience and education) – Project management (planning and follow-through) – Technical know-how	– Testing – Product management (e.g. documentation and change management – Project management (resources and milestones, customer management)
Improvement ideas	**Risks**
– Requirements management – Customer cooperation – Testing	– Personnel changes – Cost and schedule overruns – Unsatisfied customers because of defects, schedule and cost overruns

Fig. 1. SWOT-chart of the tSoft companies

In the KYKY assessment the strongest process areas were customer supplier processes, project management and software engineering processes. Most of the assessed practices got grade F or L in these process areas, but there were differences between companies. According to the KYKY results, the main improvement opportunities were support processes, process improvement and process oriented operation. In the above mentioned process areas most of the questions were assessed to P or N. For example in support process, improvement opportunities were noticed in documentation and release practices.

Table 1 summarizes the SPICE assessment's main findings. To preserve confidentiality, companies are referred to as Company A, B, C etc. in Table 1.

In the SPICE assessments the following key strengths and weaknesses were identified at participating companies.

Table 1. Capability levels of the companies process instances

	Company							
Process	A	B	C	D	E	F	G	H
Project management	1	1	1	1	2	1	2	2
Software requirements analysis	1	1	2	0	1	1	2	-
Configuration management	1	1	1	0	1	1	0	2
Quality management	0	0	0	0	1	0	1	2
Subcontractor management	-	1	0	-	-	-	1	-
Levels: 0 incomplete, 1 performed, 2 managed, - not assessed								

Project management process was performed at level 1 in all companies. One of the main strengths was that a project plan was generated in every assessed project. Generally the scope of work and the main achievements of the project were well defined and also a development strategy was described in the project plan. Software lifecycle model for the project was mainly iterative or incremental in nature, but also classic water fall model was used. Responsibilities of the project's tasks were normally well defined both internally and externally.

Although the project management process was performed at level 1 in all companies, some improvement opportunities were identified. Project's workload and time estimates were normally based on solely project manager's experience and no formal methods e.g. for product size estimation were used. Also project's activities and their associated tasks were often described very roughly. Generally there were no formal procedures for project's risk or quality management and project's quality and risk plans were missing in most cases. Also project's monitoring practices need more attention because of e.g. comparization between planned and used mandays or calendar time were rarely done. Communication and human resource management were among of the most mentioned improvement areas during different assessments.

Software requirements management process vary from one company to another. Three of the companies had formally defined requirement management process, but the rest of the companies gather and manage requirements more or less unsystematically. One of the companies didn't have any kind of requirements specification. Generally companies had some kind of requirement documents and especially functional requirements of the software were often defined. Requirement baseline documents were also reviewed together with the customer. However, in this process there were generally many things to improve in assessed projects except those three companies who followed their described process. First, requirements, like performance, usability and interface were rarely documented. Second, requirements change management was poorly organized in most of the companies. Often requirements document was not updated after first accepted version and sometimes incoming changes were stored only in the email or yellow notes. Third, tracebility of the requirements were also poor because of lack of documentation and tool support of requirements change management.

In *configuration management process* base practices of the companies varied mostly. The companies seemed to understand the basic activities of the configuration

management (i.e. version control, change management and release management) but the implementation of these activities were between ad hoc and performed in most of the assessed projects. Only two of the assessed process instances achieved level 1 base practices fully, four largely (just) and two of the processes were at level 0 because the results were only partially achieved. One strength of the configuration management process in most of the companies were the code version control which was handled by version control tool (e.g. cvs). However, in most of the cases, the tool was not used for the documents, which were handled manually. For change management purposes some of the companies had developed their own tool which was used for managing customer requests and defects. In all assessed projects some kind of project folders and files hierarchies was created, but the naming and the structure of the folders and files were often unsystematic. Delivery of the products was handled well in all cases.

The key weaknesses of the process were related to the following issues. First, the identification of the components which belonged under product management. In many cases it seemed that it was not clearly decided which of the work products (specifications, plans etc.) should be under version control and maintenance. Second, the instructions and guidelines for configuration management were lacking and members of the project group had shortage of information how to handle, for example release of the product. Third, the change management of different components was often poorly handled. Especially documents were seldom updated and the version history of them was lacking or defective. Also the responsibilities in the area of configuration management should have been assigned more clearly.

Results of *quality management process* were mostly at ad hoc level and base practices of level 1 only partly achieved. One of the assessed process instances achieved level 2, two achieved level 1 and the rest were at level 0. One of the assessed company has a ISO 9001 certification, but others were not largely aware about software quality aspects and there were no formal procedures for quality management. Despite of the non-systematic quality management process companies did perform informally some quality activities like testing and reviews (e.g. for code). However, these activities were not systematically planned and requirements for the quality of the work products were lacking. The following improvement opportunities were identified. These improvement opportunities do not concern the ISO9001 certified company.

At the first, companies should start to think, what quality means to them and to set general goals for quality management and assurance. After that they should think what are the quality goals for products and processes at the project level and start tailoring the quality goals according the project. To achieve this it means that companies should start the planning for quality. Also the activities of how the project will meet the required goals should be planned and implemented. E.g. checklists for different purposes and reviews at the right places of the project life cycle could help to meet goals. Well-planned and implemented testing is also one of the key activities for good product quality. After that, the continuous monitoring of quality situation should be arranged.

Subcontractor management process was assessed only in three companies and the results include only three process instances. Two of these instances achieved level 1

and one had partly achieved rating. General strengths of the process were definition of the scope of work and the evaluation and selection of the partners. The selection of the partners were often made based on former experience of partner and no formal capability evaluations were used. The needed contracts were well handled. Some risks and improvement opportunities were also identified. The systematic practices for checking and acceptance of work were lacking. For example, testing, versioning, virus control and checking against specification, could help. The practices for monitoring the subcontractors work and progress were insufficient. Progress report templates or reporting tool for this purpose could help. Also the specification of work and risk management practices should improve.

Conclusions

In this paper we have described software process assessment work and results at tSoft-project with 8 small and medium sized software organizations in Eastern Finland. We conducted software process assessments to examine the processes used by an organization to determine whether they are effective in achieving their goals and the results were used to drive process improvement activities.

Most of the current processes and practices in participating organizations are far from being well defined and systematically implemented and managed. We found out that competency of company personnel was a critical strength of the companies and therefore the personnel changes are a great risk for small organization.

Identified weaknesses were, for example, systematic project's work load and time estimation practices that were lacking, requirements change management was poorly organized and organizations were not largely aware about software quality aspects and there were no formal procedures for quality or configuration management. However organizations performed many of the base practices of the assessment model in different processes and achieved level 1 in several assessed process instances. Especially project management practices were generally handled well in assessed projects. Also functional requirements of the software system were often defined and version control tool for code was in use.

We believe that after assessments the organizations are now more aware of software process improvement topics and the work towards culture of quality has began. As a result of the assessment process, key findings were selected for improvement and constituted the basis for SPI projects of the companies. SPI projects were kicked off in companies during autumn 2003 and spring 2004. At the moment companies are implementing planned improvement projects and we have already seen the first results. We follow and support these improvement actions very closely and we hope to report the more detailed results later. We believe that this study provides an interesting insight into the state of the practice of small software enterprises in presented process areas.

References

1. Allen, P., Ramachandran, M., Abushama, H.: PRISMS: an Approach to Software Process Improvement for Small to Medium Enterprises. In proceedings of Third International Conference On Quality Software, November 6-7, Dallas, Texas (2003)
2. Brodman, J., Johnson, D.: Tailoring the CMM for Small Businessess, Small Organizations, and Small Projects, Software Process Newsletter No. 8, (1997)
3. Cater-Steel, A.: Process Improvement in Four Small Software Companies. In Proceedings of 13th Australian Software Engineering Conference (ASWEC'01), August 27 - 28, Canberra, Australia (2001)
4. CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS) Version 1.1, Software Engineering Institute, Carnegie Mellon University (2002)
5. Diaz, M., Sligo, J.: How Software Process Improvement Helped Motorola. IEEE Software, Vol 14, No. 5, (1997) 75-81
6. Dybå, T.: Factors of Software Process Improvement Success in Small and Large Organizations: an Empirical Study in the Scandinavian Context. In Proceedings of the 9th European Software Engineering Conference held jointly with 10th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Helsinki, Finland, (2003) 148 - 157
7. El Emam, K., Briand L.: Cost and Benefits of Software Process Improvement. Technical Report ISERN-97-12, International Software Engineering Research Network (1997)
8. Herbsleb, J., Carleton, A., Rozum, J., Siegel, J., Zubrow, D.: Benefits of CMM based Software Process Improvement: Initial Results. Technical Report CMU/SEI-94-TR-13, Software Engineering Institute, Pittsburgh, PA (1994)
9. Hofer, C., Software Development in Austria: Results of an Empirical Study among Small and Very Small Enterprises. In Proceedings of the 28 th Euromicro Conference (EUROMICRO'02), September 4 – 6, 2002, Dortmund, Germany,(2002) 361- 366
10. Horvat, R., Rozman, I. and Györkös, J.: Managing the Complexity of SPI in Small Companies. Software Process Improvement and Practice, Vol. 5, No. 1, (2000) 45-54
11. IEEE Software November/December 2003: The State of the Practice, Vol. 20, No. 6. (2003)
12. ISO/IEC IS 15504-2: Software Engineering - Process Assessment - Part 2: Performing An Assessment (2003)
13. ISO/IEC TR 15504-1: Information Technology - Software Process Assesment - Part 1: A Concepts and Introductory Guide. International Organisation for Standardisation (1998)
14. ISO/IEC TR 15504-5: Information Technology - Software process assessment - Part 5: An assessment model and indicator guidance. International Organisation for Standardisation (1998)
15. Kautz, K.: Software Process Improvement in Very Small Enterprises? Does it pay off? Journal of Software Process - Improvement and Practice, Special Issue on Organizational Change through Software Process Improvement, Vol. 4, No. 4, (1998) 209-226
16. Kelly, D., Culleton, B.: Process Improvement for Small Organizations. IEEE Computer, Vol. 32, No. 10, (1999) 41-47
17. Krasner, H.: Accumulating the Body of Evidence for The Payoff of Software Process Improvement. http://www.utexas.edu/coe/sqi/archive/krasner/spi.pdf (1997) (accessed 12.7.2004)
18. Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Koch, G., Saukkonen, S.: Software Process Assessment and Improvement, The BOOTSTRAP Approach. Blackwell Publishers, Oxford, UK (1994)

19. The KYKY model, Software Technology Transfer Finland, http://www.sttf.fi/ (Unpublished)
20. Laryd, A., Orci, T.: Dynamic CMM for Small Organizations. In Proceedings of the 1st Argentine Symposium on Software Engineering (ASSE 2000) (2000) 133-149
21. Moitra, D.: Managing Change for Software Process Improvement Initiatives: a Practical Experience-Based Approach. Software Process Improvement and Practice, Vol. 4, No. 4, (1998) 199-207
22. Rainer, A., Hall, T.: Key Success Factors for Implementing Software Process Improvement: A Maturity-Based Analysis. Journal of Systems and Software, Vol. 67, No. 2, (2002) 71-84
23. Saukkonen, S., Oivo, M.: Teollinen ohjelmistoprosessi: Ohjelmistoprosessin parantaminen SIPI-menetelmällä. Helsinki, TEKES, (1998) (in finnish)
24. SPICE assessment forms: Finnish Software Measurement Association, http://www.fisma-network.org/ (Unpublished)
25. tSoft-project: University of Joensuu, Finland, http://cs.joensuu.fi/tSoft/ (in finnish)
26. Stelzer, D., Mellis, W.: Success Factors of Organizational Change in Software Process Improvement. Software Process Improvement and Practice, Vol. 4, No. 4, (1998) 227-250

An Experimental Replica to Validate a Set of Metrics for Software Process Models

Félix García, Francisco Ruiz, and Mario Piattini

Alarcos Research Group, University of Castilla-La Mancha, Paseo de la Universidad, 4
13071 Ciudad Real, Spain
{Felix.Garcia,Francisco.RuizG,Mario.Piattini}@uclm.es
http://alarcos.inf-cr.uclm.es/english/

Abstract. The software process measurement plays an essential role in order to provide the quantitative basis necessary for software process improvement. Traditionally, this measurement has been focused in the project and product measurement, but nowadays software process models (SPM) are entities very relevant due to the increasing number of companies which model and manage their processes in order to reach high maturity levels. We have defined a set of metrics for software process models in order to evaluate the influence of the software process models complexity in their maintainability. These metrics are focused on the main elements included in a software process model. To demonstrate the practical utility of the metrics proposed a replica of an experiment has been achieved which has allowed us to obtain some conclusions about the influence of the metrics proposed on two sub-characteristics of the maintainability: understandability and modifiability, which besides confirm the results of a set of experiments previously performed in the context of a family of experiments.

1 Introduction

Software process improvement has obtained a great attention in companies in the last few years. Companies are becoming more and more concerned about the improvement of the quality of their processes in order to promote the final quality of the software products obtained. As a matter of fact, the continuous software process improvement is a fundamental objective for organizations that desire to reach higher levels of maturity according to standards and proposals like CMMI [14] and the ISO 15504 [10]. Within the context of these initiatives, to reach the aims of each level of maturity the processes have to be improved by understanding, controlling and applying improvement actions.

The successful management of the software process is necessary in order to satisfy the final quality, cost, and time of the marketing of the software products and to carry out this management, four key responsibilities need to be assumed [7]: Definition, Measurement, Control and Improvement of the process. Taking these responsibilities into account, it is very important to consider the measurement of software processes

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 79–90, 2004.

to establish the quantitative basis necessary for the identification of the areas which are candidate for the improvement. One of the main reasons of the growing interest in software metrics has been the perception that software metrics are necessary for software process improvement [6]. Measurement is essential for understanding, defining, managing and controlling the software development and maintenance processes [12].

In the measurement of software processes the following basic kind of entities can be identified:

- **Software Process Model.** The models constitute the starting point in order to understand and carry out the software process through its enactment in concrete projects. Software process modeling has become a very acceptable solution for treating the inherent complexity of software processes, and a great variety of modeling languages and formalities can be found in the literature, known as "Process Modeling Languages" (PML). With a software process model (SPM) the different elements related to a software process are precisely represented, without ambiguity.
- **Software Projects.** They are concrete enactments of software process models and their measurement is fundamental in order to know their performance measured mainly through schedule and cost metrics.
- **Software Products.** As a result of carried out software projects different products could be obtained and these are also candidate for the measurement. The quality of the process has to be reflected in the products obtained and for this reason it is necessary to measure the software products.

In the literature, the research on the software process measurement has been focused in the measurement of the projects and products, but explicit metrics on software process models have not been defined. Due to the importance of software process models in the improvement of software processes it is important to consider the influence on the processes of the quality of their models.

With this aim in mind, we have defined a representative set of metrics for software process models in order to evaluate the influence of the complexity in the software process models in their maintainability (see Table 1). The metrics are focused on the main elements included in a SPM and may provide the quantitative basis necessary to choose the model with the most easiness of maintenance among semantically equivalent SPM in organizations which are changing their models to improve their processes. It can greatly ease the evolution of their SPM.

The metrics defined are indicators of a software process model structural complexity, and they could be every useful as maintainability indicators, taking into account that a software process model with high degree of complexity will be much more difficult to change, and this can affect to their maintainability [3].

The metrics have been defined following the SPEM (Software Process Engineering Metamodel) terminology [15] by examining its key software process constructors, but they can be directly applied to other process modeling languages. The metrics defined are **Model Scope Metrics** (see Table 1), because they measure the structural complexity of the overall software process model.

Table 1. Model Scope Metrics

Metric	Definition
NA(PM)	Number of **Activities** of the software process model
NWP(PM)	Number of **Work Products** of the software process model
NPR(PM)	Number of **Roles** which participate in the process
NDWPIn(PM)	Number of input dependences of the **Work Products** with the **Activities** in the process
NDWPOut(PM)	Number of output dependences of the **Work Products** with the **Activities** in the process
NDWP(PM)	Number of dependences between **Work Products** and **Activities** $NDWP(PM) = NDWPIn(MP) + NDWPOut(MP)$
NDA(PM)	Number of precedence dependences between **Activities**
NCA(PM)	Activity Coupling in the process model. $NCA(PM) = \frac{NA(PM)}{NDA(PM)}$
RDWPIn(PM)	Ratio between **input dependences** of Work Products with Activities and **total number of dependences** of Work Products with Activities $RDWPIn(PM) = \frac{NDWPIn(PM)}{NDWP(PM)}$
RDWPOut(PM)	Ratio between **output dependences** of Work Products with Activities and **total number of dependences** of Work Products with Activities $RDWPOut(PM) = \frac{NDWPOut(PM)}{NDWP(PM)}$
RWPA(PM)	Ratio of **Work Products** and **Activities**. Average of the work products and the activities of the process model. $RWPA(PM) = \frac{NWP(PM)}{NA(PM)}$
RRPA(PM)	Ratio of **Process Roles** and **Activities** $RRPA(PM) = \frac{NPR(PM)}{NA(PM)}$

To demonstrate the practical utility of the metrics proposed as maintainability indicators a replica of an experiment has been carried out which has allowed us to obtain some conclusions about the influence of the metrics proposed on two sub-characteristics of the maintainability: understandability and modifiability. In the following section we present the empirical validation we have performed in the context of a family of experiments. In Section 2.1 we describe the results obtained in the previous experiments performed and then (section 2. 2) we describe with detail a replica of the last experiment. In Section 2.3 the results of the third experiment and its replica are compared. Finally, some conclusions and further works are outlined.

2 Empirical Validation of the Model Scope Metrics

In order to prove the practical utility of the metrics it is necessary to run out empirical studies. In this section we describe an experiment (replica of a previous experiment) we have performed to empirically validate the proposed measures as early maintainability indicators. This experiment is part of a family of experiments we are carrying

out, according to the guidelines provided in [2], in order to establish if there is relationship between the metrics proposed at model scope (which evaluates the structural complexity of SPM) and the SPM maintainability measured through the understandability, modifiability and analysability (dependent variables).

2.1 Previous Experiments

In the experiments previously performed in the context of the family of experiments we provided the subjects 18 SPM and they rated their maintainability. These experiments can be classified in the following two groups:

- **Subjective Experiments**. In these two experiments the subjects (students, researchers and assistant professors) rated each of the three maintainability sub-characteristics according to a scale composed of seven linguistic labels (from extremely easy to extremely difficult for each sub-characteristic). The results obtained in these experiments [8] reflect that several of the model level metrics (NA, NWP, NDWPIn, NDWPOut, NDWP y NDA) were highly related to software process models maintainability.
- **Objective Experiment**. Even though the results of the subjective experiments were good, we were aware that the way of measuring the dependent variables was subjective and relied solely on judgment of the users, which may have biased the results. Therefore, we decided to carry out another experiment [9] in which the subjects were professionals of a software company and we measured the dependent variable in a more objective way. In this experiment the dependent variables considered were the understandability and the modifiability of the SPM. To measure these variables the subjects had to answer five questions and perform four modifications on the models. We obtained the time the subjects spent answering the questions (understandability time) and the time subjects spent carrying out the tasks required (modifiability time) in order to demonstrate if there was relationship between the metrics and the understandability and modifiability time. As a result of this experiment we could validate some metrics (NA, NWP, NDWPIn, NDWPOut, NDWP and NDA) respect to the understandability time, but we could not demonstrate any relationship between the metrics and the modifiability time. We think these results were produced because the subjects – before performing the modifications – had previously answered the questions related with the understandability. This fact was taken into account in the planning of the 4th Experiment.

2.2 Description of the 4th Experiment

The experiment described in this section is the fourth experiment of the family and it is a replica of the last one. To carry out this experiment we have followed some suggestions provided in [16][11] [4] and [5] on how to perform controlled experiments. In the following subsections are described the results obtained according to the format proposed in [16].

2.2.1 Definition

Using the GQM template [1], for goal definition, the experiment goal is defined as follows:

Analyse	*Software process models (SPM) structural complexity metrics*
For the purpose of	*Evaluating*
With respect to	*their capability of being used as software process model maintainability indicators*
From the point of view of	*Researchers*
In the context of	*Students of third course of Software Engineering*

2.2.2 Planning

The planning was carried out according to the following steps:

- **Context selection.** The context of the experiment is a group of undergraduate students and hence the experiment is run off-line (not in an industrial software development environment). The subjects were two groups of students enrolled at the Department of Computer Science at the University of Castilla-La Mancha in Spain. The first group was composed of forty-six students enrolled in the final-year (third) of the Computer Science (BSc) in the speciality of Management and the second group were forty-one students enrolled in the final-year in the Systems speciality of the Computer Science (BSc). The experiment is specific since it is focused on SPM structural complexity metrics. The ability to generalize from this specific context is further elaborated below when discussing threats to the experiment. The experiment addresses a real problem, i.e., what indicators can be used for the maintainability of SPM? With this end in view it investigates the correlation between SPM structural complexity metrics and two maintainability sub-characteristics (understandability and modificability).
- **Selection of subjects.** The subjects have been chosen for convenience, i.e., the subjects are students who have experience and knowledge in software product modelling (UML, databases, etc.), but they have not experience or knowledge in the conceptual modelling of SPM.
- **Variables selection.** The independent variable is the SPM structural complexity. The dependent variables are the understandability and the modifiability.
- **Instrumentation.** The objects have been 10 SPM belonging to different standards and methodologies. These models are a representative subset of the 18 original models provided in the previous experiments of the family which were selected to avoid the fatigue effects because in this experiment the subjects were students. To select this representative subset we considered for each model its structural complexity (metrics values) and its understandability time obtained in the previous experiment. The independent variable has been measured through the metrics proposed at model scope (see Table 1). The dependent variables have been measured by the time the subjects spent in carrying out the exercises included in each model, which could be one of the following: answering five questions related with the understandability of the model (understandability time) or carrying out four modifications activities on the model (modifiability time) (see Appendix A). Our assump-

tion here is that, the faster a class diagram can be understood and modified, the easier it is to maintain.

- **Hypotheses formulation.** We wish to test the following two set of hypotheses:
  1) Null hypothesis, H_{0e}: There is no significant correlation between structural complexity metrics and the understandability time.
  2) Alternative hypothesis, H_{1e}: There is significant correlation between structural complexity metrics and the understandability time.
  3) Null hypothesis, H_{0m}: There is no significant correlation between structural complexity metrics and the modifiability time.
  4) Alternative hypothesis, H_{1m}: There is significant correlation between structural complexity metrics and the modifiability time.
- **Experiment design.** We selected a within-subject design experiment, i.e., all the tests (experimental tasks) have had to be solved by each of the subjects. The subjects were given the tests in different order and the ten models selected respect to the original material [9] were grouped in the following way: Group X: Models 1, 2, 3, 9 and 10; Group Y: Models 4, 6, 7, 12 and 17.

  For each model two exercises sheets were prepared: one in which it was required to answer the understandability questions (X_u, Y_u) and another one containing the modification exercises (X_m, Y_m). To distribute the material, and considering that we had to provide ten SPM for each subject (five with understandability exercises and five with modifiability exercises) we decided to provide to the subjects of the Management Group the material packages X_u, Y_m and to the subjects of the Systems Group the material packages Y_u and X_m.

2.2.3 Operation

It is in this phase where measurements are collected, including the following activities:

- **Preparation.** The experiment took place in the same day but in different timetable for each group of subjects (management and systems). Subjects were given an intensive training session before the experiment took place. However, subjects were not aware of what aspects we intended to study. Neither were they aware of the actual hypotheses stated. We prepared the material handed to the subjects consisting of ten SPM and one example solved. These models were related with different universes of discourse but they were general enough to be understood by the subjects. The structural complexity of each diagram is different, because as table 2 shows, the values of the metrics are different for each model.

  Each model had an enclosed test (see appendix A) that included one of the following two sections: the first composed of five questions related with the model and the second composed of different steps to perform for the modification of the model. Depending on the model and the group of subjects (management or systems), each subject had to answer the questions or perform the modifications specified. The modifications to each SPM were similar, including adding and deleting of activities, work products, roles and their dependences.

Table 2. Metric values for each model

Model	NA	NWP	NPR	NDWPIn	NDWPOut	NDWP	NDA	NCA	RDWPIn	RDWPOut	RWPA	RRPA
1	6	6	3	5	6	11	6	1.000	0.455	0.545	1.000	0.500
2	5	6	4	5	5	10	4	1.250	0.500	0.500	1.200	0.800
3	2	13	2	12	3	15	1	2.000	0.800	0.200	6.500	1.000
4	9	25	9	25	21	46	11	0.818	0.543	0.457	2.778	1.000
6	4	11	4	14	9	23	3	1.333	0.609	0.391	2.750	1.000
7	8	17	1	15	11	26	9	0.889	0.577	0.423	2.125	0.125
9	7	12	1	12	11	23	6	1.167	0.522	0.478	1.714	0.143
10	24	37	10	72	40	112	24	1.000	0.643	0.357	1.542	0.417
12	2	8	3	6	4	10	1	2.000	0.600	0.400	4.000	1.500
17	4	24	1	20	11	31	3	1.333	0.645	0.355	6.000	0.250

- **Execution.** The subjects were given all the materials described in the previous paragraph. We explained how to do the tests. We allowed one hour and a half to carry out the experiment (we previously performed a pilot experiment to determine the average duration). We collected all the data consisting of the times of understanding and modification, the understandability answers and the modifications performed on the models.
- **Data Validation.** Once the data was collected, we have controlled if the tests were complete and if the modifications had been done correctly. We discarded the tests of one subject in the management group because the times in three models were missing. Therefore, we have taken into account the responses of 45 subjects in the group of management and 41 subjects in the group of systems.

2.2.4 Analysis and Interpretation

We had the metric values calculated for each SPM (see table 2), and we have calculated the mean of the understandability and modifiability time. So these are the data we want to analyse to test the hypotheses stated above. We have applied the Kolmogorov-Smirnov test to ascertain if the distribution of the data collected was normal. As the data were non-normal we have decided to use a non-parametric test like Spearman's correlation coefficient, with a level of significance $\alpha = 0.05$, correlating each of the metrics separately with understandability time and with modifiability time (table 3).

For a sample size of 10 (mean values for each diagram) and $\alpha = 0.05$, the Spearman cutoff for accepting H_{0e} and H_{0m} is 0,6320 [17]. Because the computed Spearman's correlation coefficients for the understandability time (see table 3) for the metrics NA, NWP, NDWPIn, NDWPOut, NDWP, NDA and NCA are above the cutoff, and the p-value < 0,05, the null hypothesis H_{0e}, is rejected. Hence, we can conclude that there is a significant correlation between these metrics and the understandability time. Respect to modifiability time there is relationship between the metrics NA,

NWP, NDWPIn, NDWPOut and NDWP and the mean time required to perform the modifications required on the models.

Table 3. Spearman's correlation coefficients between metrics and understandability and modifiability time

Metric	Spearman'correlation coefficients understandability time	Spearman'correlation coefficients modifiability time
NA(PM)	**0.841** p=0.002	**0.640** p=0.046
NWP(PM)	**0.826** P=0.003	**0.650** p=0.042
NPR(PM)	0.074 p= 0.838	0.377 p=0.283
NDWPIn(PM)	**0.786** p=0.007	**0.738** p=0.015
NDWPOut(PM)	**0.886** p=0.001	**0.791** p=0.006
NDWP(PM)	**0.893** p=0.001	**0.707** p=0.022
NDA(PM)	**0.821** p=0.003	0.599 p=0.067
NCA(PM)	**-0.752** p=0.012	-0.44 p=0.203
RDWPIn(PM)	0.79 p=0.828	0.115 p=0.751
RDWPOut(PM)	-0.79 p=0.828	-0.115 p=0.751
RWPA(PM)	-0.116 p=0.751	-0.30 p=0.934
RRPA(PM)	-0.560 p=0.092	-0.141 p=0.697

2.2.5 Validity Evaluation

We will discuss the various issues that threaten the validity of the empirical study and how we attempted to alleviate them:

- **Threats to Conclusion Validity.** The size of the sample data (860 values, 10 models and 86 subjects) constitutes a significant size that allow us to obtain a conclusion validity.
- **Threats to Construct Validity.** The dependent variables we used are understandability and modifiability time, so we consider these variables constructively valid.
- **Threats to Internal Validity.** Seeing the results of the experiment we can conclude that exists empirical evidence of relationship between the independent and the dependent variables. We have tackled different aspects that could threaten the internal validity of the study, such as: differences among subjects, knowledge of the universe of discourse among SPM, precision in the time values, learning effects, fatigue effects, persistence effects and subject motivation. With respect to the previous experiment with professionals, in this experiment we specially focused in the subjects motivation by giving them a special lecture about software process modeling and technology and we reduced the average duration of the experiment to avoid fatigue effects.
- **Threats to External Validity.** The following threats to external validity that have been identified which could limit the realism of the experiment [13]:
  - Materials and tasks used. In the experiment, we have used software process models based on standards and methodologies found in the bibliography and tasks representative of real cases, but more empirical studies, using real software process models from software companies, must be carried out.

- Subjects. The experiment has been performed by students and for this reason we could not generalize the results in the same way that in the previous experiment with professionals. However, the results obtained in this experiment has confirmed the results obtained in the experiment with professionals.
- Environment. The experiment was done by using pen and paper. In future experiments we could consider the use of software tools to perform the activities required in order to provide a more realistic environment.

2.3 Comparison of Results

In table 4 we summarize the results we have obtained from the third experiment and its replica in order to select the metrics that influences in the understandability (Und) and modifiability (Mod) of SPM:

Table 4. Metrics validated in the 3rd and 4th experiments

		NA	NWP	NDWPIn	NDWPOut	NDWP	NDA	NCA
3rd Exp	**Und**	✓	✓	✓	✓	✓	✓	
	Mod							
4th Exp (replica)	**Und**	✓	✓	✓	✓	✓	✓	✓
	Mod	✓	✓	✓	✓	✓		

As we can observe in table 4, the results of the replica confirm the results obtained in the previous experiment because the metrics NA, NWP, NDWPIn, NDWPOut, NDWP and NDA are related to the understandability. Besides, with the carrying out of the replica it has been possible to demonstrate the relationship between some metrics (NA, NWP, NDWPIn, NDWPOut and NDWP) and the modifiability. These results confirm our conclusion at the end of the third experiment that the understandability tasks had influence in the time used by the professionals to perform the modifications required [9]. Also, as a result of the replica seems that there could be a relationship between the metric NCA and the understandability. It is necessary to confirm this in future studies.

3 Conclusions and Future Work

Software process measurement plays an essential role in order to provide the quantitative basis necessary for software process improvement. Traditionally, this measurement has been focused in project and product measurement, but nowadays the software process models (SPM) are entities very relevant due to the increasing number of companies which model and manage their processes in order to reach higher maturity levels.

In this work we have proposed a set of representative metrics to provide a quantitative basis to evaluate the influence of the SPM complexity in the quality of the process. These metrics are based on the main elements included in a SPM and may be

very useful to evaluate software processes evolution in companies with high maturity levels.

In order to evaluate the relationship between the structural complexity of the software process models, measured through the metrics proposed, and their maintainability we have carried out a replica of an experiment to select the metrics related with the time necessary to understand and modify a software process model. This replica has allowed us to confirm some conclusions about the influence of the metrics NA, NWP, NDWPIn, NDWPOut, NDWP, NDA and NCA as good understandability indicators and the metrics NA, NWP, NDWPIn, NDWPOut and NDWP as good modifiability indicators. These results confirm the results obtained in the experiments previously performed in the context of a family of experiments.

Although the results obtained are encouraging, we cannot consider them definitive results. It is necessary elaborate new experiments centered in the evaluation of concrete metrics we consider relevant like the metrics NCA and NPR. According to the issues previously identified, we can point out the following lines for improvement in future studies:

– Carrying out new experiments focused on the evaluation of concrete metrics we consider relevant (NPR, NCA) and that according the experiments of the family performed until now seem not to be clearly correlated with the maintainability of software process models.
– Carrying out case studies using real software process models.
– Consideration of other views related with the modeling of software processes, like for example roles and their responsibilities on work products, in order to define and validate new possible metrics.

Acknowledgements

This work has been funded by the MAS project partially supported by "Dirección General de Investigación of the Ministerio de Ciencia y Tecnología" (TIC 2003-02737-C02-02).

Appendix A

Management Group:

SPM 1. (see Figure 1). Answer the following questions:

Write down the starting hour (indicating hh:mm:ss): ___________

1. Can the Technical Designer **Define the User Interface**? ____
2. Is it possible to initiate the activity **Refine the User Interface** before the activity **Define the User Interface**? ____
3. Is it necessary to use the product *User Work Processes* for the activity **Refine the User Interface**? ____
4. Is the product *User Interface (refined)* output of the activity **Design Process Model**?____

5. When the activity **Refine User Interface** is carried out, have the *Technical Requirements* been produced? ____

Write down the ending hour (indicating hh:mm:ss): ____________

Systems Group:

SPM 1. (see Figure 1). Carry out the necessary modifications to satisfy the following requirements:

Write down the starting hour (indicating hh:mm:ss): ___________

1. It is necessary to use the product *Requirements Preliminary Information* for the execution of the activity **Define Requirements.**
2. It is not necessary to finish the activity **Define Requirements** to start the **Definition of the User Interface**, but it is necessary the **Definition of the User Interface** to be executed after the activity **Design the Process Model**.
3. It is desired to include the new activity **Final Review**, after the **Building of the Application.** This new activity receives as input the *Application* and produces an *Approval Document.*
4. The Programmer is responsible of the **Final Review** and also participates in the Building of the Application.

Write down the ending hour (indicating hh:mm:ss): ___________

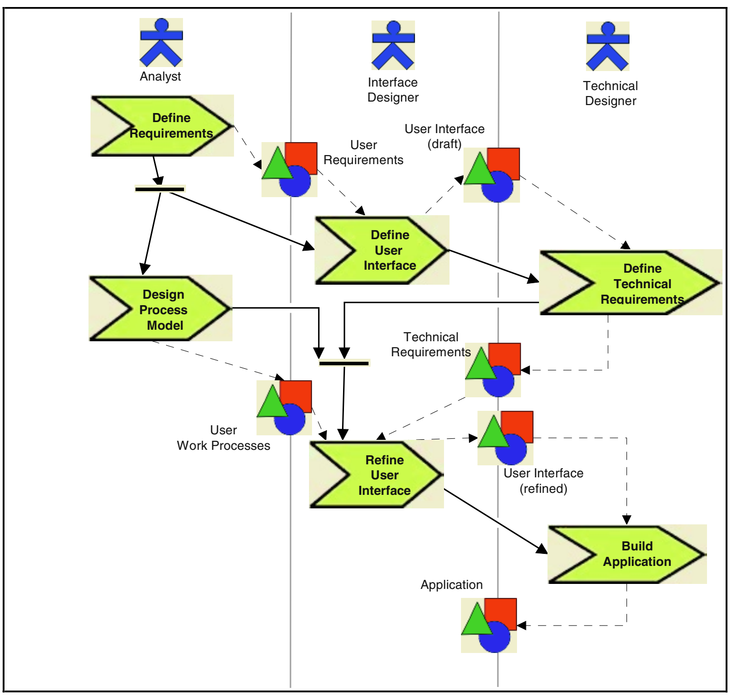

Fig. 1. Example of a Software Process Model represented with SPEM

References

1. Basili, V. and Rombach H. The TAME project: towards improvement-oriented software environments. IEEE Transactions on Software Engineering, 14(6), (1988), 728-738.
2. Basili, V., Shull, F. and Lanubile, F. Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering, 25(4), (1999), 435-437.
3. Briand., L., Wüst, J. and Lounis, H. A. Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study. In Technical Report ISERN-98-29, International Software Engineering Research Network (1998).
4. Briand L., Arisholm S., Counsell F., Houdek F. and Thévenod-Fosse P. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions. Empirical Software Engineering, 4(4), (1999), 387-404.
5. Calero, C., Piattini, M. and Genero, M.: Empirical Validation of referential metrics. Information Software and Technology". Special Issue on Controlled Experiments in Software Technology. Vol.43, Nº 15 (2001).
6. Fenton, N.: Metrics for Software Process Improvement. Software Process Improvement: Metrics, Measurement and Process Modelling. Haug, M., Olsen, E. W. and Bergman, L (eds). Springer (2001), 34-55.
7. Florac, W. A. and Carleton, A.D. Measuring the Software Process. Statistical Process Control for Software Process Improvement. SEI Series in Software Engineering. Addison Wesley (1999).
8. García, F., Ruiz, F. and Piattini, M. Proposal of Metrics for Software Process Models. *Software Measurement European Forum (SMEF 2004)*, Rome, 28-30 January (2004), 406-416.
9. García, F., Ruiz, F. and Piattini, M. Definition and Empirical Validation of Metrics for Software Process Models. *Proceedings of the 5th International Conference Product Focused Software Process Improvement (PROFES'2004)*. Lecture Notes in Computer Science (LNCS 3009). Kansai Science City (Japan), April (2004), 146-158.
10. ISO/IEC: ISO IEC 15504 TR2:1998, part 2: A reference model for processes and process capability, (1998).
11. Perry, D., Porte, A. and Votta, L. Empirical Studies os Software En-gineering: A Roadmap. Future of Software Engineering. Ed:Anthony Finkelstein, ACM, (2000), 345-355.
12. Pfleeger, S.L.: Integrating Process and Measurement. In Software Measurement. A. Melton (ed). London. International Thomson Computer Press (1996) 53-74.
13. Sjoberg, D., Anda, B., Arisholm, E., Dyba, T., Jorgensen, M., Karahasanovic, A., Koren, E. and Vokác, M. Conducting Realistic Experiments in Software Engineering. Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02).
14. Software Engineering Institute (SEI). Capability Maturity Model Integration (CMMI^{SM}), version 1.1. March (2002). In http://www.sei.cmu/cmmi/cmmi.html
15. Software Process Engineering Metamodel Specification; adopted specification, version 1.0. Object Management Group. November (2002). Available in http://cgi.omg.org/cgi-bin/doc?ptc/02-05-03.
16. Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B. and Wesslén A. Ex-perimentation in Software Engineering: An Introduction, Kluwer Academic Publishers. (2000).
17. http://department.obg.cuhk.edu.hk/researchsupport/Minimum_correlation.asp

Using Measurement Data in a TSP^{SM} Project

Noopur Davis[1], Julia Mullaney[1], and David Carrington[2]

[1] Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
[2] School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, QLD 4072, Australia
davec@itee.uq.edu.au

Abstract. One of the challenges for software engineering is collecting meaningful data from industrial projects. Software process improvement depends on measurement to provide baseline status and confirming evidence of the effect of process changes. Without data, any conclusions rely on intuition and guessing. The Team Software Process^{SM} (TSP^{SM}) provides a powerful framework for data collection and analysis, in addition to its primary goal as a basis for highly effective software development. In this paper, we describe the experiences of, and benefits realized by, a team using the TSP for the first time. By reviewing how this particular team collected and used data, we show features of the TSP that make it a powerful foundation for software process improvement.

1 Introduction

The Capability Maturity Model® for Software (SW-CMM®) [7] is one of the earliest and most influential models for software process improvement. Because it focuses on the organizational level and describes "what" rather than "how", the model can prove challenging to apply, particularly to small organizations or projects. Humphrey [3] developed the Personal Software Process^{SM} (PSP^{SM}) using all the applicable SW-CMM practices up through level 5 to demonstrate how the principles are just as applicable to individual software engineers.

Humphrey then developed the Team Software Process (TSP) [1, 4, 5] to build and sustain effective teams using the PSP. TSP creates self-directed teams with a defined process and responsibility for improving it. It also provides a practical measurement framework. For most organizations, TSP "out of the box" is a significant improvement over their current practice, but it incorporates a continuous improvement philosophy that encourages tailoring based on personal and team experience.

Our goal for this paper is to demonstrate how the TSP provides a framework for data collection and analysis for industrial software engineering projects. To do this, we describe a project at a multi-national company who worked with the Software Engineering Institute to pilot test the TSP. As we describe the project from start to completion, we explain what the TSP is, and the types of data that are collected and

^{SM} Personal Software Process, PSP, Team Software Process and TSP are service marks of Carnegie Mellon University.

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 91–101, 2004.

analyzed in a TSP project. This particular project was chosen because complete project data exist, from the initial TSP launch through to project completion. This project also illustrates typical problems faced by software projects and how the TSP data help the decision making required to overcome such problems. A more complete description of the project is available in [1], which also contains results from multiple TSP projects from multiple organizations.

The paper is organized as follows: Section 2 provides background information on the PSP and the TSP, Section 3 describes the TSP launch for the project while Section 4 describes the evolution of the project through to its completion. Section 5 reflects on the project outcomes by looking at both quantitative and qualitative information. Section 6 concludes the paper by reviewing how the TSP can be used as a foundation for software process improvement and empirical software engineering case studies.

2 Overview of the PSP and the TSP

Most software engineers do not plan and track their work, nor do they measure and manage product quality. This is not surprising, since typically engineers are neither trained in these disciplines nor required to use them. The dilemma is that until they try using disciplined methods, most software engineers do not believe that these methods will work for them. They will not try these methods without evidence, and they cannot get the evidence without trying the methods. This dilemma applies to software process improvement in general [9].

The PSP addresses this dilemma by putting engineers in a course environment to learn the methods. The engineers use the methods in the course and can see from their personal and class data that the methods can and do work for them. In this way, the PSP is software process improvement at the individual level. The positive impact of the PSP has been analyzed by Hayes and Over [2] and replicated by Wesslén [11]. The objective of the TSP is to provide a team environment that supports PSP work and to build and maintain a self-directed team.

2.1 The PSP

The PSP course is composed of ten programming assignments and five reports. The PSP methods are introduced in six upwardly compatible steps in which the engineers write one or two programs at each step, gathering and analyzing data on their work. They use their data and analyses to improve their work.

The PSP planning and quality principles [4] are:

- Every engineer is different; to be most effective, engineers must plan their work and they must base their plans on personal data.
- To improve their performance consistently, engineers must measure their work and use their results to improve.
- To produce quality products, engineers must feel personally responsible for the quality of their products. Superior products are not produced by accident; engineers must strive to do quality work.
- It costs less to find and fix defects earlier in a process than later.
- It is more efficient to prevent defects than to find and fix them.
- The right way is always the fastest and cheapest way to do a job.

Using the PSP, engineers collect three basic measures: size, time, and defects. For the purposes of the PSP course, size is measured in lines of code (LOC). In practice, engineers use a size measure appropriate to the programming language and environment they are using; for example, number of database objects, number of use cases, number of classes, etc. To ensure that size is measured consistently, counting and coding standards are defined and used by each engineer. Derived measures that involve size, such as productivity or defect density, use new and changed LOC only. "New and changed LOC" is defined as lines of code that are added or modified; existing LOC is not included in the measure. Time is measured as the direct hours spent on each task and does not include interrupt time. A defect is anything that detracts from the program's ability to meet the users' needs completely and effectively. A defect is an objective measure that engineers can identify, describe, and count.

Many other measures are derived from these three basic measures. Both planned and actual data for all measures are gathered and recorded. Actual data are used to track and predict schedule and quality status. All data are archived to provide a personal historical repository for improving estimation accuracy and product quality.

2.2 The TSP

Prechelt and Unger [8] observe that PSP training does not ensure that engineers apply these methods to their work. The TSP aims to encourage adoption by establishing the necessary peer encouragement. The TSP is based on the following principles:

- The engineers know the most about the job and can make the best plans.
- When engineers plan their own work, they are committed to the plan.
- Precise project tracking requires detailed plans and accurate data.
- Only the people doing the work can collect precise and accurate data.
- To minimize cycle time, the engineers must balance their workload.
- To maximize productivity, focus first on quality.

The TSP has two primary components: a team-building component and a team-working or management component. The team-building component of the TSP is the TSP launch, which puts the team in the challenging situation of developing their plan.

The TSP Launch. A TSP launch is the first step in a project. The team, led by a qualified TSP coach, reaches a common understanding of the work and the approach they will take, produces a detailed plan to guide the work, and obtains management support for their plan. A launch is composed of nine meetings over four days as shown in Fig. 1.

Meeting 1 helps the team understand what they are being asked to do. Marketing (or a customer representative) and management meet with the team and describe their product needs and the business needs respectively, as well as any constraints under which the team will have to work. In the next seven meetings, the team develops an engineering plan to meet the business and product needs before reporting.

In meeting 2, the team organizes itself and its goals. Team members take on team roles to create individual responsibility and interdependence within the team. In meeting 3, the team determines its overall project strategy based on a conceptual design. In meeting 4, the overall team plan is developed by estimating the size of the various products to be produced, identifying the tasks to be performed, and estimating the required effort. The plan for the next development cycle (typically 2-3 months) is

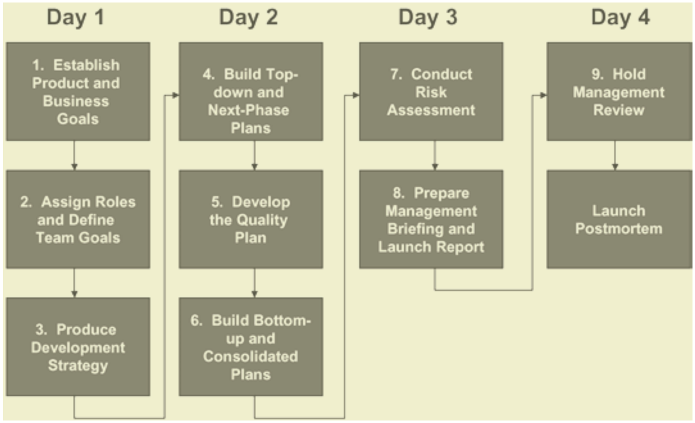

Fig. 1. The structure of the TSP launch

Fig. 2. The TSP launch products

elaborated to tasks of about ten hours or less. A schedule is created of the team's available work hours week by week for the complete project.

In meeting 5, the team defines a plan to meet their quality goals by estimating the number of defects injected and removed in each phase and calculating the defect density in the final product. For teams new to the TSP, parameters for the quality plan can be derived either from the team's PSP data or from the TSP guidelines provided by the SEI.

In meeting 6, tasks in the next development cycle plan are allocated to team members. Each team member creates an individual plan based on their own historical data and available hours per week for this project. The team reviews the individual plans to ensure that the workload is balanced. Then the individual plans are consolidated into the team plan that is used to guide and track work during the next development cycle.

In meeting 7, project risks are identified and their likelihood and impact are assessed. In meeting 8, the team prepares a presentation of the team's plan for management. If the team plan does not meet management's goals, the team includes alternative plans derived, for example, by adding resources or reducing functionality.

In meeting 9, the team presents its plan to management for approval. The alternative plans provide management with options and their consequences in cases where the primary plan does not meet the business needs. At the end of a TSP launch, the team and management agree on how the team will proceed. The team has a plan it believes in and can track against. The launch not only creates a plan (see Fig. 2 for the plan components), it also builds a cohesive team.

After completion of major project milestones, or whenever the team finds that its existing plan no longer guides its work, the team uses the TSP relaunch to develop a revised plan for the next set of detailed near-term tasks for the project. A TSP relaunch has a similar structure to the original launch but without meetings 1, 8 and 9. A relaunch is a convenient event for introducing new members to a project team.

After the Launch. The TSP includes guidance for ensuring that the energy and commitment generated by a TSP launch are sustained as the team does its work. A TSP coach works with the team and the team leader to help the team collect and analyze data, follow the process defined by the team, track issues and risks, maintain the plan, track progress against goals (especially the team's quality goals), and report status to management. The TSP uses the same basic measures of the PSP – size, time, and defects – and adds task completion dates. For all measures, planned and actual data are collected at the individual level. The TSP measurement framework consolidates individual data into a team perspective. The data collected are analyzed weekly by the team to understand project status against schedule and quality goals. The TSP measurement framework also provides other views of the data, such as by sub-product or sub-part, by phase, by task, by week, by day, by individual, or by any set of individuals on a team. Personal and team data are archived to provide a repository of historical data for future planning purposes.

The team conducts weekly meetings to report progress against their plans and to discuss team issues. They also use their TSP data to make accurate status reports to management on a regular basis. Because management can rely on the data, their job changes from continuously checking project status to ensuring that there are no obstacles impeding the team's progress. Management is also able to make sound business decisions, based on accurate engineering data. For example, when management is confident in the team's estimate, management can decide how to allocate resources to obtain a schedule that best meets the business needs. When a team commitment is in jeopardy, the team solves the problem or raises the issue with management as early as possible. In all cases and at all levels, decisions are made based on data.

3 The Project: Launch

Less than one month after completing the PSP for engineers training, the first TSP project in this company was launched. The team consisted of five team members, including the team leader. The team leader was open-minded about the TSP and fully supported its use. Of the other four team members, one had fully embraced the PSP, another felt comfortable using disciplined processes, and the other two were skeptical.

At the first launch meeting, senior management explained the importance of the product to the business and their expectation that the software be delivered to the testing group in nine months (36 weeks). In the same meeting, a marketing representative discussed the product needs with the team. The subsequent meetings proceeded mostly as expected. There was some impatience to get to what the team considered

"real planning" or who does what when. However, they agreed to follow the TSP process. The conceptual design stage took extra time to complete as the team wanted to go into more detail than necessary at this point of the launch.

The plan developed by the team (Table 1) showed software delivery to the test group eleven weeks after management's schedule goal. Some team members were concerned about giving bad news to management so the launch coach asked each team member if they thought the plan was too conservative. Every team member agreed with the 47 week schedule.

In meeting 9, the team leader presented the plan to management who asked questions and expressed concern about the planned task hours per team member per week. After discussions, management accepted the team plan and challenged the team to improve its planned task hours per week.

Feedback from the team at the launch post-mortem included remarks about how much they had achieved in three days and how the launch process had guided them step by step. Team members would have liked more time prior to the launch to work on a conceptual design and more time to do size estimating in meeting 4. The TSP launch coach as a neutral facilitator was also seen as a positive influence.

Table 1. Project Plan Summary

Delivery to testing group	Week 47
Ready to release	Week 58
Effort estimate	2814 hours
New and changed LOC	14.5 KLOC
System test defect density	0.36 defects/KLOC
Average task hours per team member per week	15 task hours/week

4 The Project: Executing the Plan

Like most software projects, this team encountered obstacles as it started executing its plan: some risks were realized, some tasks were underestimated, and requirements changed and grew. Snapshots of the project are presented at:

- Week 13, just prior to the first relaunch,
- Week 46, prior to the original date the team committed to deliver the software to the test group,
- Week 49, the week the team delivered the software to test, and
- Week 62, the week the product was ready for release.

4.1 Week 13

During the launch, the team identified as a high impact risk the possibility that excessive work on legacy products might reduce available hours to work on the new product. As shown in Table 2, this risk was realized. By Week 13, the team was about 15% behind in task hours (planned 650 hours vs. actual 565.1 hours) and had underestimated the work by 38% (plan of 306.6 hours for work completed vs. actual 497.8 hours). They had earned only half of the planned earned value. If the project continued at the current rate, the schedule exposure was at least six months.

Table 2. Team status at Week 13

	Plan	Actual	Plan/Actual
Project hours to date	650.0	565.1	1.15
To-date hours for tasks completed	306.6	497.8	0.62
Earned value to date	46.9	23.8	1.97

In response to this data, management added three new team members to the project at the relaunch in Week 15. Based on lessons learned from the first fourteen weeks and the new team members, the team created a new plan with an effort estimate of 3328 hours (15% increase) but an unchanged release date (Week 58).

4.2 Week 46

Team data at Week 46 (Table 3) shows the original commitment to deliver the product to the test group was 3% behind schedule (96.5 planned vs. 93.6 earned).

Table 3. Team status at Week 46

	Plan	Actual	Plan/Actual
Project hours to date	3319.2	3829.8	0.87
To-date hours for tasks completed	3051.7	3715.9	0.82
Earned value to date	96.5	93.6	1.03

The earned value chart (Fig. 3) predicted completion by Week 49, two weeks behind the date committed to at the original launch, rather than the six months predicted in Week 13.

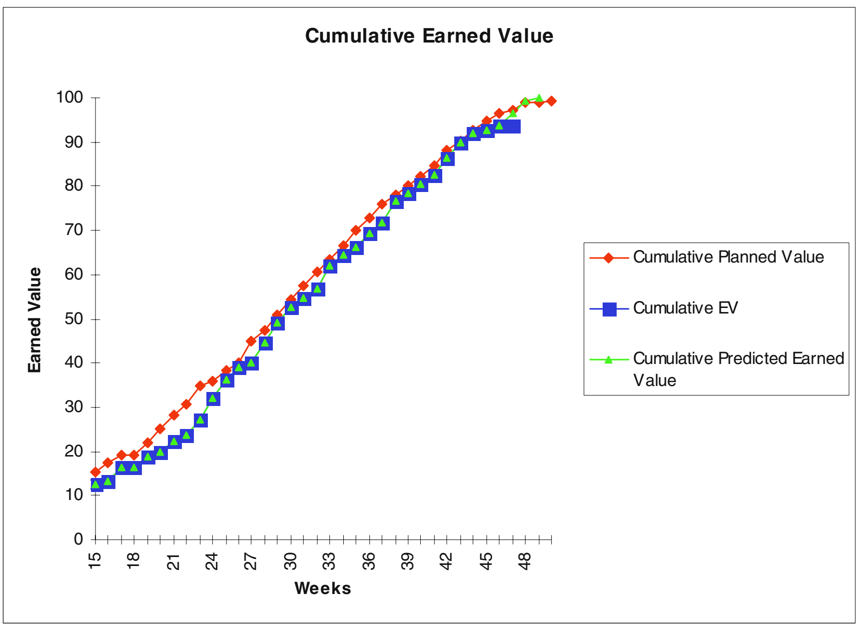

Fig. 3. Cumulative earned value through Week 46

This improvement was achieved through:

- adding three new team members and a co-op student (on placement in industry),
- continual monitoring of status and taking corrective action as required, and
- improving average task hours per person as shown in Fig. 4 (Week 18 was the Christmas week).

The task hour improvement was not achieved by accident or by working overtime; rather it was planned during the relaunch and by increasing uninterrupted time on task by adopting a quiet time policy, streamlining necessary meetings, and eliminating unnecessary ones. Through task hour management, team productivity increased by 28% (Table 4).

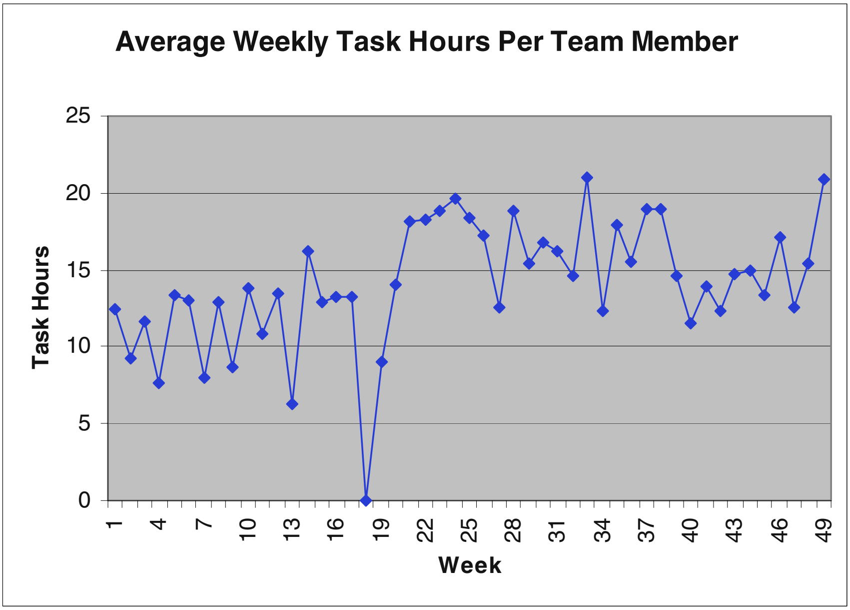

Fig. 4. Average weekly task hours per team member

Table 4. Average task hours per engineer per week

Before relaunch (weeks 1-14)	11.36
After relaunch (weeks 15-49)	15.77

4.3 Week 49

The team delivered the software to the testing group in Week 49. About a week prior to this, the team was asked to implement an additional requirement. The team estimated that the new requirement would take 252 task hours. Management did not want to affect the release date of Week 58. Using their historical data, the team believed they could develop the additional functionality while the testing group tested the initial functionality, since they were confident the project was of high quality and they expected to spend little time fixing defects found in system test.

4.4 Week 62

As with typical software projects, testing proved to be an unpredictable activity. Even though very few defects were found during system test, one defect was found only in a “last-minute” regression test. The product was released in Week 62, four weeks beyond the date the team had committed to over a year earlier, and with more functionality than originally planned. The final project status is shown in Table 5. Compared to a previous release of a similar product, these results represent:

- a ten times reduction in the number of problems logged by the testing group,
- an eight times reduction in system test duration.

Table 5. Final project status – plan vs. actual

	Plan	Actual
Delivery to testing group	Week 47	Week 49
Product ready to release	Week 58	Week 62
Effort estimate (not including functionality added in Week 48)	2814 hrs	3670 hrs
New and Changed LOC (KLOC)	14.5	28.9
System test defect density (defects/KLOC)	0.36	0.44
Average task hours per team member per week	15	14.5

5 Project Reflections

This was a typical project in terms of size and expected duration for this part of the organization. The original criterion for success was to meet the ship date, which this project missed by four weeks, a 7% deviation. However, the ship date was slipped with full knowledge of and approval from management, because the team was able to keep management informed of the latest project status at all times. Also, the team delivered additional functionality. It is important to realise that the criteria for success normally changes as a project progresses: as the team and the organization learn more, requirements change, other unforeseen events occur. As long as the change is managed and expectations adjusted in line with the changes, the project meets its success criteria.

This project illustrates many problems faced by software projects: initial estimates wrong, requirements change and grow, and the team continually interrupted with work that is not directly related to the project. It is impossible to identify any one thing as “this is why the project succeeded”. Certainly, the team has to know as early as possible when there is a problem. TSP teams collect data daily and review schedule and quality status weekly. This allowed this team to recognize in the early weeks of the project that they were falling behind schedule.

In addition, the team must understand what is causing the problem. For this team, the work required was underestimated and the available task hours overestimated. The project data allowed management to make a decision to allocate extra resources to the project and the team to change its work practices to increase their actual task hours.

Another contributor to the team's success was its focus on quality, which was not abandoned under schedule pressure. By actively managing quality, the team produced a high quality product and did not get into repeated test-and-fix cycles. The team

spent more than one third of their total development effort on quality enhancing activities such as design, personal reviews, and team inspections (Fig. 5). Prior to system testing, 945 defects were removed, leaving less than 0.44 defects/KLOC to be found in system testing.

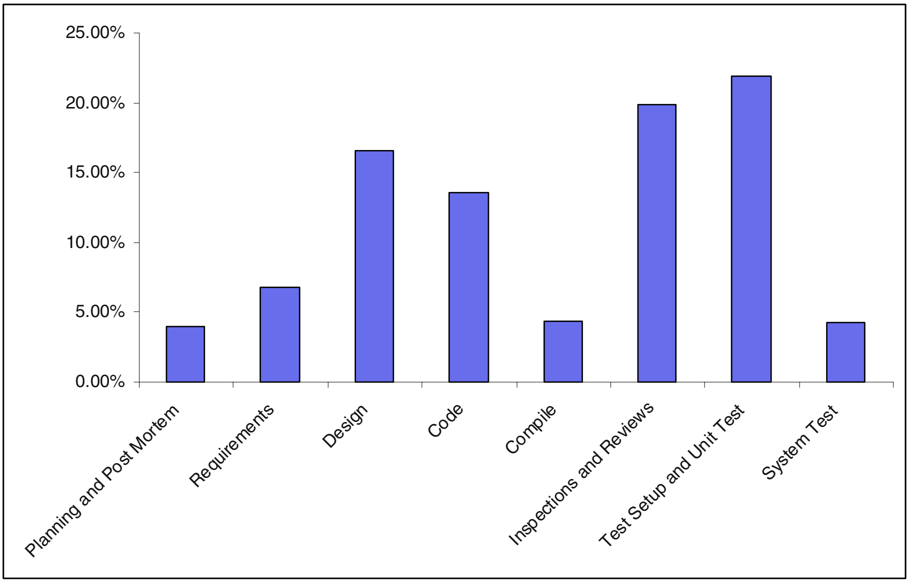

Fig. 5. Effort distribution

While the importance of disciplined methods cannot be underestimated, the intangible human elements of a team also determine whether a project succeeds or fails. The launch process helps the team establish shared goals, identify the interdependencies between team members, build a believable plan, and make commitments explicit. The TSP project process helps the team track their work relative to their plan, and adapt as circumstances change. All team members said that they enjoyed working on this project. The manager of the testing group said it was one of the most stable releases that his group had ever tested.

6 Conclusions

For organizations that develop software, the TSP offers a quantitative approach to managing their software projects. Based on experiences to date [1, 6], significantly improved software quality and ability to meet schedule are also likely outcomes. TSP is a structured approach to organizing software projects, but it strongly encourages adaptation by the project team as circumstances change or the team identifies new methods of working. This is consistent with the concept of high maturity in software processes where continuous process improvement is enabled by quantitative feedback and from piloting innovative ideas and technologies.

For software engineering researchers, the TSP provides a valuable base for industrial experimentation. Generally, empirical studies of industrial software projects are

considered difficult and expensive to arrange [10]. The TSP shows that collecting and analyzing data need not be an overhead; rather it can add to the effectiveness of the software project team by focusing their attention on what is really happening and allowing them to adapt in a timely manner. If a project is using the TSP, the impact of new techniques or tools can be assessed much more easily than in the typical environment where no data is available on prior performance. Standard TSP data about the product (size and defects by phase of injection and removal) and the process (time by phase) allow numerous hypotheses to be explored. The TSP also offers the significant advantage of consistency of data definition and collection, which assists with replication of experiments.

Acknowledgements

The authors would like to thank the members of the TSP Initiative team at the SEI, in particular, Jim Over, the team leader and Watts Humphrey who created both the PSP and the TSP.

References

1. Noopur Davis and Julia Mullaney. The Team Software Process (TSP) in Practice: A Summary of Recent Results, Technical Report CMU/SEI-2003-TR-014, 2003.
2. Will Hayes and James Over. The Personal Software Process: An Empirical Study of the Impacts of PSP on Individual Engineers, Technical Report CMU/SEI-97-TR-001, 1997.
3. Watts S. Humphrey. A Discipline for Software Engineering, Addison-Wesley, 1995.
4. Watts S. Humphrey. The Team Software Process (TSP), Technical Report CMU/SEI-2000-TR-023, 2000.
5. Watts S. Humphrey. Winning with Software: An Executive Strategy, Addison-Wesley, 2002.
6. Donald McAndrews. The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices, Technical Report CMU/SEI-2000-TR-015, 2000.
7. Mark C. Paulk, Charles V. Weber, Bill Curtis, Mary Beth Chrissis. The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, 1995.
8. Lutz Prechelt and Barbara Unger. An experiment measuring the effects of personal software process (PSP) training, IEEE Transactions on Software Engineering, 27(5):465-472, 2001.
9. Austen Rainer, Tracey Hall and Nathan Baddoo. Persuading Developers to 'Buy into' Software Process Improvement: Local Opinion and Empirical Evidence, Proc. International Symposium on Empirical Software Engineering, pp.326-335, IEEE Computer Society, 2003.
10. Dag I.K. Sjøberg et al. Conducting Realistic Experiments in Software Engineering, Proc. International Symposium on Empirical Software Engineering, pp.17-26, IEEE Computer Society, 2002.
11. Anders Wesslén. A Replicated Empirical Study of the Impact of the Methods of the PSP on Individual Engineers, Empirical Software Engineering: An International Journal, 5(2):93-123, 2000.

Software Thinking Improvement Learning Performance Improving Lessons

Keld Pedersen

Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, DK-9220 Aalborg, Denmark

Abstract. SPI paradigm is dominated by improvement based on quantitative process metrics. The assumption behind this research is that improvement should be based, not just on insight in quantitative data about development processes, but also on insight in how developers think about system development: Software is developed by people, and it is the developers' perceptions, experience and thinking about system development that guides their behavior. The present research develops an approach based on causal maps and counterfactual thinking that supports developers in learning from individual system development projects. The research adds to the body of knowledge concerning management and learning in system development practice.

Keywords: System Development, Project Management, Organizational Learning, Reflection.

1 Introduction

Software Process Improvement (SPI) is concerned with improving general development and management processes. Organisations are advised to establish processes that incorporate best practices within software engineering, and to improve these processes incrementally using quantitative data about productivity and quality (e.g. Humphrey, 1989). The present research address two closely related concerns that are neglected by the mainstream SPI literature. First, collecting data about what system developers *think* about system development practice is important because developers behavior are guided by their perceptions of, experience with and thinking about system development. Schoen (1983) argue that practitioners solve problems primarily by relying on personal experience. System developers acquire knowledge about how to practice by encountering certain types of situations again and again, and by building a repertoire of examples, images, understandings and actions. New situations are perceived as unique but at the same time seen as something already present in the developers repertoire of past experience, and *seeing* as leads to *doing* as. It is the professionals' reflective capacity to see unfamiliar situations as familiar ones, and to do in the latter as they have done in the former that enables them to exploit past experience. Guindon, Krasner and Curtis (1987) has described how experienced designers uses their repertoire of past design-schemas and adapts the most relevant to the needs of the present situation to come up with solutions.

Second, improving developers' reflective skills is important because system development is not accomplished just about following pre-defined processes or best practices, and because performance-improving lessons do not emerge just by gathering

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 102–113, 2004.

quantitative process data. According to Schoen (1983) professional practice depends on individuals reflective skills to define the problems that needs to be solved, solve the problems, deal with unexpected events and to correct over-learning. Over-learning happens when development practices are stable and becomes tacit, taken for granted, and beyond discussion. Lyytinen and Robey (1999) argues that system developers not only are bad at learning from past failures, they actually learn to fail: System developers internalise practices that in many cases leads to unsatisfactory results. Through reflection system developers can attempt to surface and evaluate these tacit understandings and practices and create suggestions for improvement. There is little support for this kind of reflection in the system development literature, and very little room for this kind of reflection in system development practice. To change this situation two issues needs further attention:

- How to support system development practitioners in explicating how they understand key issues in their own system development practice.
- How to use this understanding to come up with new alternative practices that might improve the situation.

These two issues are addressed in this research. Section 2 describes the research approach, section 3 describes a new approach that system development practitioners can use to learn about and improve development practice, and section 4 describes the experiences from using the approach. Section 5 outlines the further implications.

2 Research Approach

The research builds upon literature studies, lab and field experiments (Gill and Johnson, 1997). The literature study identified relevant contributions within related areas. Based on these contributions an approach to support learning from system development projects was defined, tried and refined during lab experiments involving 40 university students working with small development projects (5–7 persons working together for 4 month). Finally the approach was tried and further refined trough field experiments involving two projects groups (4–6 persons) in a small system development organisation. The organisation had initiated a in SPI project, and the field experiments took place within the context of the overall SPI project. During these experiments data was collected by having participants document individually lessons learned about the system development events under investigation both *before* and *after* using the approach, by video taping the sessions and by interviewing the participants about the approach a few days after the sessions. Each session took 4 hours. The participants pre and post lessons made it possible to evaluate if the sessions made any difference. The data collected by videotaping the sessions was used to identify problems and opportunities for improvement, while the data from the interviews was used to collect the participants' qualitative experiences with the approach.

3 Practice Learning

The approach suggested here is a four step process that can be used e.g. as part of a post mortem review in a system development project: Setting up the context, Select-

ing an issue, Building a map and Learning from the map. The steps are described in the following subsections.

3.1 Setting up the Context

Successful learning in development organisations depends on the context. Organisations must provide the needed resources (e.g. people, time, money, facilitator skills, processes and guidance), valid and timely feedback about development practice, they should design the specific learning practice in a participatory manner to get commitment from developers, establish a blame-free atmosphere were information about mistakes and failures can be shared, and they should integrated new learning practices with the existing ways of learning and knowledge sharing in the organisation (Pedersen 2004). Furthermore it is important to create realistic expectations. Learning from one's own development practice is quite challenging, and the best outcome is reached if the effort is guided by specific learning goals, if there are established ways to ensure that lessons learned are shared and exploited, and if resources are used on understanding the issues with the highest degree of relevance for future development projects in the organization (Pedersen 2004).

3.2 Selecting an Issue

Each of the participants list issues from the project that they would like to learn from. Based on the list the participants choose one or two issues that have a high degree of relevance for future development efforts. Everybody should feel comfortable taking part in the investigation, and participants are allowed to reject issues they find unpleasant to discus in public.

3.3 Building a Causal Map

The purpose is to surface the basic assumptions that shape development practices. One way of doing this is to use causal maps. Causal maps have been used in different contexts to explicate how people claim to perceive causal relationships. Recently causal maps has been used to explicate tacit knowledge (Ambrosini and Bowman, 2001) and to build an understanding of software operations support expertise (Nelson et. al., 2000). Other kinds of maps (Lanzara and Mathiassen, 1985) has been proposed to support reflection within system development as well. Traditionally causal maps have been used to study the belief system of politicians and managers (Nelson et. al., 2000). The "real" causal maps or theories-in-use, that guide developers behaviour are only accessible to the degree that developers are able and willing to share them with other people. What we are able to map is therefore called *revealed* causal maps and contain causal assertions *as expressed by the participants*. A context that is perceived to be unsafe by the participants, or where competition, political games, being polite etc is more important than honest communication will refrain the participants from expressing what they believe to be true.

The starting point is the issue selected in the previous step. In this step the participants builds their own collective theory that explains *why* this specific issue or event turned out the way it did. The theory is modelled as a revealed causal map using a simple graphical notation.

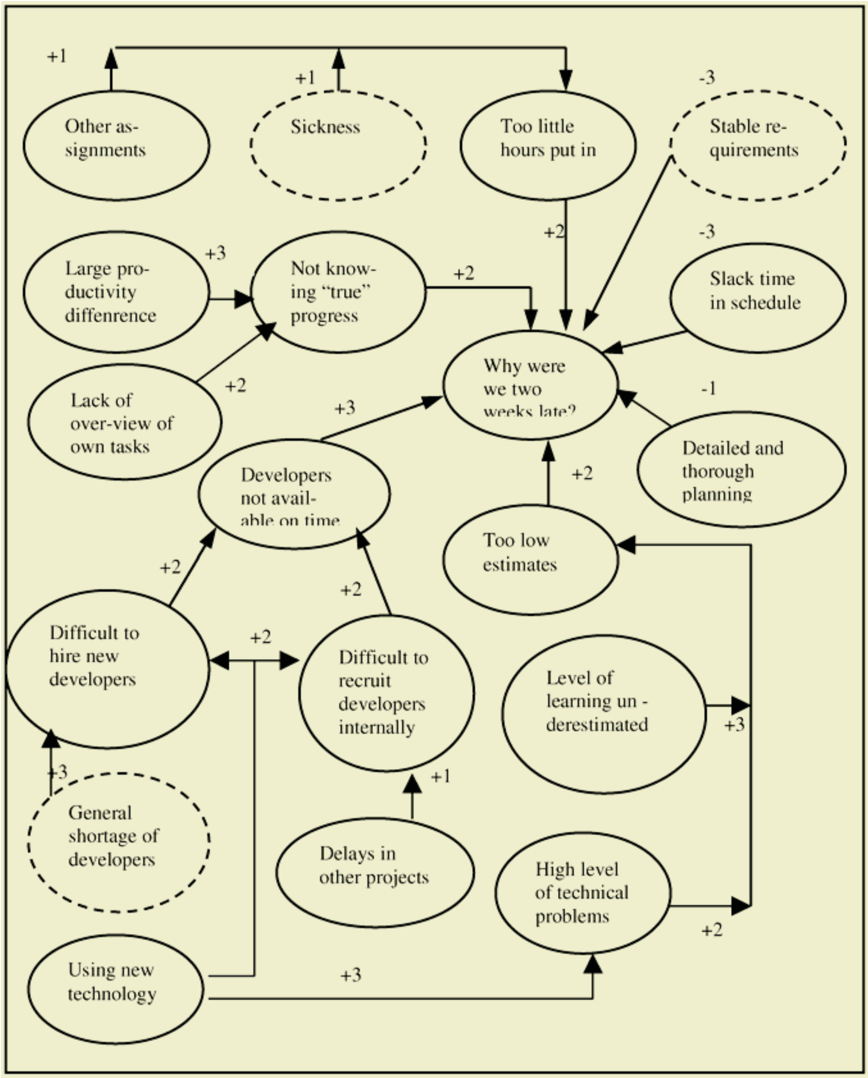

Fig. 1. Causal map

Figure 1 contains parts of a map constructed by the author during an investigation of a development project he took part in. An oval illustrates a cause or an effect and an arrow illustrates a causal relationship. The starting point is a specific question like “Why was the first construction phase in our project 2 weeks delayed?” When trying to answer this question a map is drawn that illustrates the cause-effect relationships as perceived by the participants, focusing on the actions, events, conditions etc that shaped the outcome of the task. By systematically adding further detail to the map, investigating the “causes-of-causes” a richer understanding of the situation emerges.

The causal map is validated using available quantitative and qualitative data, and the different perceptions of the problem as experienced by the participants. For each

causal relationship, we ask ourselves why we believe this to be true. Some issues are easy to settle. In figure 1 one of the causes is “Stable requirements”. By studying the project data about the number of requirements changes, it was easy to verify that this was actually the case. Other relationships are more difficult to validate. “Lack of overview of own tasks”, meaning that one of the reasons for not “knowing true progress” was that individual developers lacked overview of the tasks they were supposed to do, and therefore rather late in the development process discovered that they were seriously behind schedule, was more difficult to verify.

Different biases makes it difficult for us to draw valid lessons when something goes wrong. The biases depend on our position in the organisation. Managers confronted with poor subordinate performance, are biased for attributing poor performance to group members, whereas group members are likely to attribute it to external or situational factors (Hofman and Stetzer, 1998). At the individual level there is a tendency to overlook failures. Individuals are more biased for attributing their successes to their own excellence, and their failures to bad luck than the other way around (Levinthal and March, 1993).

Two factors are important determinants for how much energy the participants should devote to specific parts of the map: The *perceived impact* of a causal relationship, and the *perceived probability* that the causal relationship will occur again. High impact / high probability causes are more interesting than low impact / low probability causes.

To visualise the perceived impact of a relationship positive or negative numbers can be added to the causal connectors, using 3 to denote a strong relationship and 1 to denote a weak relationship. By adding plus or minus signs in front of the numbers it is illustrated whether the impact of the cause is evaluated to be positive or negative to the effect. Evaluating the strength of the causal relationships is not an exact science, but just a simple way to focus the remaining part of the investigation.

To visualize the probability that a specific cause will happen again the ovals are shown with dotted lines (low probability) and solid lines (high probability).

3.4 Learning from the Map

The completion of the previous step results in a revealed causal map that express how the participants explain the issue under investigation. This map is valuable in itself, e.g. in the example in figure 1, the map can be used when planning future projects. The theory visualised in the map is insufficient from a positivist scientific perspective. It is not general, it only includes what the participants choose to reveal, the participants might be biased by different interests, historically revisionism etc., and it is biased towards unexpected and surprising causes. However the map can still serve as a vehicle for learning. It can be used to elicit basic assumptions that are invalid because reality has changed, and it can be used to generate performance-improving lessons.

3.4.1 Challenging the Basics: Why Did It Surprise Us?

Making a complete map covering all causes for even a simple event in a system development project is impossible, and participants are likely to emphasise the causes that surprised them. Investigating the causes that caught the participants by surprise is

interesting and carries valuable lessons learned *because they often express that some deeply routed assumptions about the development practice turned out to be wrong*.

In the example map in figure 1, the two causes "Large difference in individual productivity" and "Lack of overview of own tasks" express that the project manager relied on old management traditions in the company that was outdated. For many years the development organisation had hired developers with an academic education and the potential to become project managers. The unspoken expectation was that all developers were able to manage a development project. For several reasons the company started hiring people with different backgrounds, but without project managers fully understanding the management related implications.

3.4.2 Identifying New Ways of Working: How Can We Get a Better Result?

Another way to learn from the map is to use it to support systematic counterfactual thinking (CFT) – wishful thinking about how things might have turned out differently (Morris and Moore, 2000). CFT happens after the fact and works by having us consider the factors (conditions, events, actions etc.) that are temporally antecedent to the outcome and have impact on the outcome. From the counterfactual thoughts lessons are constructed linking a change in one or more factors to a difference in outcome, this way serving as a lesson to be used in future similar situations. This kind of reflection can promote performance-improving lessons if focused in the right direction, but it also offers the tempting opportunity to attribute the disappointing outcome to some external factor far from ourselves and out of our control, thereby avoiding blame and guilt (Morris and Moore, 2000). Figure 2 uses the example from last section to illustrate the different categories as defined by Morris and Moore (2000). The vertical dimension illustrates that CFT can be *upward* or *downward*.

Upward CFT compares the actual outcome with a better outcome, and focuses on how a better result could have been achieved by changing one or more of the antecedent factors (Nasco and Marsh, 1999; Morris and Moore, 2000).

Downward CFT compares the actual outcome with a worse outcome, and focus on what changes in antecedent factors that could have lead to this even worse outcome. Downward comparisons increases our satisfaction with the actual outcome because it show us that it could have gone even worse, while upward counterfactual comparisons help us draw performance improving lessons for the future (Morris and Moore, 2000). Self-efficacy – a person's expectations of being able to successfully implement the selected actions – has impact on a person's feelings about upward and downward CFT. Low efficacy and upward CFT makes persons feel worse, than high efficacy and upward CFT (Sanna 1997). The horizontal dimension illustrates that our counterfactual thoughts can be focused on our *self* or on *others*. Self-focused CFT is about what the individual person, or team, doing the thinking could have done that would have resulted in a different outcome, while other-focused CFT is about what other persons, systems, organisations etc. could have done that would have resulted in a different outcome. In general other focused comparisons do not promote performance-improving lessons for the individual team, because these comparisons naturally leads the team to focus on factors they have limited influence on. On the other hand, when we use self-focused CFT we are more likely to generate a performance improving lessons because we focus on the factors that we actually are able to change, and by engaging in this kind of thinking we also develop productive intentions about what to do differently in the future.

Of the four combinations upward and self-focused CFT provides the best opportunities to generate performance-improving lessons for the future (Morris and Moore, 2000).

The previously constructed causal map can support the process of stepping through the causal chain in order to generate alternative realities where the real outcome is changed by mentally changing antecedent factors. In figure 2 a single cause from the causal map in figure 1 (Developers not available on time) has been subject to counterfactual thoughts, but this could have been done for all the causes leading to a list of lessons that could be used to improve performance in future similar situations. The focus should be on the causes that have the biggest impact and highest probability to occur again.

	Self focused	**Other focused**
Upward	*CFT*: If we had been more aware of the other projects and their problems, we would have made a different and more flexible project plan.	*CFT*: If the development manager had kept the other projects under control, we would have got new people on time.
Downward	*CFT*: If we had been less thorough doing detailed planning and tracking, we would have been even more delayed.	*CFT*: If the development manager had not reacted fast by hiring new developers when the other projects failed to release people as originally planned, we would have been even more delayed.

Fig. 2. Four categories of CFT

There is a problem though: Upward self-focused CFT is inhibited by the high degree of accountability to hierarchical managers that characterises many organisations (Morris and Moore, 2000). If we are kept responsible for failures, are punished, or suspect that we might be punished, for failures, it is unlikely that we will engage in upward self-focused CFT because these thoughts are self-critical and put us in a vulnerable position. Even doing it in private is inhibited because we, with out being aware, are severely constrained by the surrounding social context (Morris and Moore, 2000). Furthermore engaging in upward CFT has emotional cost, and can lead to distress, dissatisfaction, disappointment and self-blame (Nasco and Marsh, 1999).

The bottom line is that self-focused upward CFT leads to intentions to change and, in some cases, to actual improvement (Morris and Moore, 2000), but also that this kind of thinking is inhibited by a high degree of accountability to hierarchical managers, and that it can make us feel less comfortable with ourselves.

4 Experiences so Far

The approach has been used to facilitate learning processes in development projects conducted by experienced system developers as described in section 2. Given the limited number of experiments it makes no sense to use advanced statistically methods to evaluate the results and because it is *field* experiments a lot of factors were not controllable, and it is impossible to accurately determine what actually caused the effect. However, the sessions resulted in a significant number of new lessons learned,

the project members refined and elaborated existing lessons learned based on discussions with the rest of the project group, and some existing lessons learned were rejected. There was a difference between the lessons formally agreed upon in public, and what the participants noted in the individual responses to what they themselves had learned from taking part in the investigation. People draw different lessons from the same events, and creating a shared understanding is not accomplished by using 4 hours to investigate a problem.

Besides measuring whether the sessions lead to new lessons learned, the participants was interviewed to collect data about the perceived usefulness of the approach. The manager in charge would have preferred to use *less* time on mapping and *more* time on generating counterfactual lessons. He liked the idea of generating performance-improving lessons that were relatively easy to implement because they focused on what the people involved themselves could do differently. However the ordinary developers involved appreciated the opportunity to improve their understanding of events they had struggled with during the development projects.

The following sections report the results from using the approach in two system development projects. Compared to the minimal resources required and the simplicity of the approach the experience indicates that it is clearly worthwhile.

4.1 The Customer Didn't Understand Our Design

The first field experiment involved a project that recently had delivered an upgrade of a standard application to a foreign customer. The project group had worked hard on improving the analysis and design process and the resulting specifications, but the delivered system didn't comply with the customers needs. From the project group's point of view the problem was that "The customer didn't understand the design document". In order to understand the problem and improve future development practice the project group developed a causal map trying to identify the causes that created this problem, and engaged in systematic counterfactual thinking about what to do differently in the future. A total of 32 contributing causes were identified and related in a causal map. The mapping process involved a lot of discussions among the participants, and these discussions helped identify major break downs in terms of basic assumptions about development practice that turned out to be false. The lessons stated here are the participant's lessons, not the researchers, but all participants did not share all lessons.

4.1.1 Specifications, Specifications and Specifications

The participants' initial perception were that there was one best practice for analysis and design work: Disciplined use of detailed specifications. Having specifications was necessary for contractual and technical reasons, and the participants had not considered whether other approaches would be better in the situation at hand, or whether the inevitable specifications should be supplemented by prototypes. Year's back the organization had improved by enforcing a strict specification practice, but the implementation was almost too successful in the sense, that it now was difficult to suggest other practices. During the investigation all the participants acknowledge that using prototypes would have reduced some of the problems, but more important that the project group needed to define method selection as a explicit activity in future pro-

jects in order to better adapt the development process to the characteristics of individual projects, and that the organisation needed to establish new principles, standards and methods, and to invest in further education in analysis and design work.

4.1.2 It Is Sufficient to Get the System Requirements

The organisation delivered individual solutions based on a standard application also developed by the organisation. As a consequence many of the customer's requirements were formulated as low-level technical changes to the existing standard system e.g. "We need two more fields – x and y - on window z". One of the consequences of communicating about requirements this way was that the developers involved never got a deeper understanding of why the requirements were relevant, how important they were, or how the customers work processes actually took place, and this lack off knowledge made it more difficult for the developers to help the customer getting the requirements right. During the investigation the participants acknowledged that they needed to get a deeper understanding of the customer organisation, and to document this knowledge e.g. as workflow descriptions in order to share it with other developers, and in order to make it the consequences of the design more transparent for the users.

4.1.3 Standards Can Secure a Uniform Level of Quality

The organisation had invested resources in establishing design standards to ensure a uniform and consistent design across the system and to support inexperienced developers. Even though standards were necessary, the project also realised that standards couldn't compensate for differences in individual skills and knowledge. In order to deal with the situation the organisation could either accept that developers had different competencies and assign them to the tasks that best suited these competencies, or improve their competencies through existing initiatives e.g. placing them at a customer site to see how typical customers worked or new initiatives e.g. courses in analysis and design work.

4.1.4 If It Passes the Internal Review It Must Be OK

As part of the project the developers implemented a review process with internal participants. During the investigation the participants acknowledge that the reviews in some cases had resulted in a false feedback about the quality of the design, because the reviewers didn't have sufficient understanding of the customers needs and work processes. The reviewers focused on the issues they were knowledgeable about: technical issues and compliance with standards. What the participants learned during the investigation was that a design review is meaningless if the participants don't have the proper background.

4.1.5 If the Customer Accepts the Document It Must Be OK

During the design phase the design documents were reviewed and approved by the customer. These reviews contributed to the overall quality of the design, but still too many design errors slipped through. According to the team the major causes were that the document was written in too technical terms, didn't communicate the design in ways that were understood by users, the users trusted the developers and if something was hard to understand the reaction was "even if I don't understand it, it is probably

ok", the design document was written in English (which were neither the first language of the developers nor the users), and finally the users didn't differentiate between what had been discussed and what was actually put into the design document. Specifications would always be necessary for contractual reasons, but the project acknowledged that the situation could be improved, by improving developers' skills in writing with the users in mind, and by training users in reading and understanding design documents, and in taking part in future design processes.

4.2 Why Don't We Succed in Knowledge Sharing?

The second field experiment involved a project in the same organization. The participants wanted to improve their learning and knowledge sharing capabilities. The project was the first project to use a specific technology, they wanted to share their experience with other projects, but felt that their effort to do so had failed. In order to get a deeper understanding of this a causal map including 24 causes was developed and discussed. The data from the session indicates that the participants learned quite different lessons, and little consensus was achieved.

4.2.1 If Nobody Reads the Standard, It's a Waste of Time

As part of the effort to share knowledge about how to use the new technology one of the team members developed a standard to be used by future projects. The project group's perception was initially that doing so had been a waste of resources because nobody seemed to read the standard. Performing a deeper inquiry revealed that people were actually using the *knowledge* in the standard, they just didn't do it by reading the document. They used the knowledge by asking and seeking help from the developer who wrote the standard, and by looking at the programs he developed using the standard. Writing a standard could rightly be perceived as a way to legitimise using time on thinking and refining past experiences, and the standard was valuable as a body of knowledge possessed by the expert, but valueless as a document, because nobody read it. Writing the standard made the expert an even more qualified expert.

4.2.2 Knowledge Is Only Properly Shared Through Documents and the Formal Systems

When examining the causal map developed by the team it turned out that they had actually succeeded in spreading a lot of knowledge about their experiences throughout the small development organization, *but they had never perceived it as knowledge sharing*. In their perception knowledge sharing was mostly about publishing official documents and presenting their findings at department meetings, and not about the informal and almost invisible knowledge sharing that took part as an integrated part of being a member of the organization. During the investigation several informal ways of knowledge sharing was recognized e.g.: Socialising, discussing problems during lunch, physical proximity, collaborative problem solving, participation in more than one project, helping each other dealing with technical problems, copying other developers code etc. Actually the organization had a quite efficient informal system for knowledge sharing, while the more formal and planned approaches like department meetings, standards, presentations etc were less successful.

The two field experiments here illustrate the kind of lessons that can be expected when using the approach: It is an investigation of how the participants think about development practice, and what consequences this thinking have for their actions, not an investigation of general processes.

5 Implications

For more than a decade frameworks for understanding and improving system development practice have been dominated by *quantitative* approaches and by a strong focus on *processes*. Within system development research it is generally recognised that both quantitative and qualitative research methods are needed to understand all aspects of development practice (e.g. Mingers, 2001), and that the outcome of system development projects depends heavily on the individual developers (e.g. Boehm, 1981). The way developers *work* depends on how they *think* about system development. This indicates that mainstream SPI frameworks might benefit from adopting a more balanced focus including qualitative approaches that focus on developers, how they think about system development and the basic assumptions that shape their development practice. The present research contributes to system development practice and research by suggesting such an approach. The preliminary results from using the approach indicates that it is possible to surface some of the basic assumptions that shape development practice by using causal maps, and that these maps serves as an excellent starting point for systematic and productive counterfactual thinking. Besides offering some specific techniques, the research has some more general implications:

- Organisations engaged in SPI can benefit from a balanced focus that includes collecting qualitative data about how developers think about development.
- Improvement requires that individual developers think different and act different. In order to achieve this developers' basic assumptions about system development, their *theories-in-use*, must be made explicit and subject for discussion.
- Lessons learned are not automatically performance improving. In general upward and self focused lessons are the most productive, and we are most likely to produce such lessons when the context is perceived to be psychological safe.

References

1. Ambrosini and Bowman (2001) Tacit Knowledge: Some Suggestions for Operationalization, Journal of Management Studies, Vol. 38 (6).
2. Boehm (1981) Software Engineering Economics, Englewood Cliffs, N. J.: Prentice Hall.
3. Gill and Johnson (1997) Research Methods for Managers, Second edition, Paul Chapman Publishing, London, England.
4. Guindon, Krasner and Curtis (1987), Breakdowns and Processes during the early activities of Software design by professionals, Empirical studies of programmers: Second workshop (Norwood, N.J.: Ablex Publishing Corp.
5. Hofmanm and Stetzer (1998) The role of safety climate and communication in accident interpretation: Implications for learning from negative events, Academy of Management Journal, Vol. 41(6).
6. Humphrey (1989) Managing the Software Process, SEI Series in Software Engineering, Addison-Wesley.

7. Lanzara and Mathiassen (1985) Mapping Situations Within a Systems Development Project, Information and Management, 8(2).
8. Levinthal and March (1993) The Myopia of Learning, Strategic Management Journal, Vol. 14.
9. Lyytinen and Robey (1999) Learning failure in information systems development, Information Systems Journal, 9(2).
10. Mingers (2001) Combining IS Research Methods: Towards a Pluralist Methodology, Information Systems Research, Vol. 12(3).
11. Morris and Moore (2000) The Lessons We (Don't) Learn: Counterfactual Thinking and Organizational Accountability After a Close Call, Administrative Science Quarterly, Vol. 45.
12. Nasco and Marsh (1999) Gaining Control through Counterfactual thinking, Personality and Social Physiology Bulletin, Vol. 25(5).
13. Nelson, Nadkarni, Narayanan and Ghods (2000) Understanding Software Operations Support Expertise: A revealed Causal Mapping Approach, MIS Quarterly, Vol 24(3).
14. Pedersen (2004) Barriers for Post Mortem Evaluations in System Development, Proceedings from UKAIS 2004.
15. Sanna (1997) Self-Efficacy and Counterfactual Thinking: Up a Creek With and Without a Paddle, Personality and Social Psychology Bulletin, Vol. 23(6).
16. Schon (1983) The Reflective Practitioner – How Professionals Think in Action. New York: Basic Books.

The Adoption of an Electronic Process Guide in a Company with Voluntary Use

Nils Brede Moe and Tore Dybå

SINTEF ICT, NO-7465 Trondheim, Norway
{nils.b.moe,tore.dyba}@sintef.no
http://www.sintef.no

Abstract. For process models to be useful, increasingly more software companies not only tailor their process models to the specific needs of the company, but also make them available on the company's intranet. In this paper, we try to understand what characterizes the use of an electronic process guide among developers in a medium sized Software House. We describe the findings that emerged from a grounded theory study. The results show that implementing an electronic process guide is not a straight forward process. The insights gained from this study can be used as recommendations for other medium sized Software Houses when implementing electronic process guides.

1 Introduction

Effectively disseminating process knowledge to process participants is crucial in any software process improvement (SPI) effort. Process participants need effective guidance when process conformance is important, when a process changes frequently, and when new personnel join a project.

Traditionally, this has been the realm of large organizations, and the way of describing and communicating processes has focused on printed standards and handbooks. However, such handbooks are often of limited use as SPI facilitators, and especially so in small and medium-sized companies.

For process models to be useful, increasingly more software companies not only tailor their process models to the specific needs of the company, but also make them available on the company's intranet. This way the traditional process handbook shifts from a bulky pile of paper to a flexible on-line structure allowing easy access to all relevant information by means of an electronic process guide (EPG) [6], [9].

An EPG can be seen as a structured, workflow-oriented, reference document for a particular process, and exists to support participants in carrying out the intended process [5]. A process guide typically includes the following basic elements:

- *Activities*: descriptions of "how things are done", including an overview of the activities and details regarding each individual activity.
- *Artifacts*: details regarding the products created or modified by an activity, either as a final or intermediate result of the activity or as a temporary result created by one of the steps.
- *Roles*: details regarding the roles and agents involved in performing the activities.
- *Tools and Techniques*: details regarding the tools and techniques used to support or automate the performance of an activity.

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 114–125, 2004.

Based on these elements, Kellner et al. [5] have proposed a set of basic requirements and design principles for EPGs. Most importantly, an EPG should provide all the information elements and relationships contained in a good paper-based process guide. In addition, it should capitalize on diagrams, tables, and narrative to provide an effective user interface. Also, it should make extensive use of hyper-links to support flexible navigation and direct access to supporting information such as examples and templates.

However, the potential of EPG's can only be realized when key capabilities are not only adopted, but also infused across the organization. This dichotomy between system availability and system use has been noted by Fichman and Kemerer [4], who distinguished between a firm's adoption of a technology and its assimilation of it. At the individual level, there is also a growing body of studies focusing on the determinants of technology acceptance and utilization (e.g. [3], [7], [11]).

New information technologies, such as an EPG, represent innovations for the potential adopters. Consequently, much of the research on individual adoption of information technologies derives its roots from the diffusion of innovation literature, in which individuals' perceived characteristics of innovating (PCI), among other factors, are posited to be significant influences on user acceptance ([7], [8]).

Other models that attempt to explain the relationship between user perceptions, attitudes, use intentions, and eventual system use include the technology acceptance model (TAM) [3], [11], the theory of planned behavior (TPB) [1], and the model of personal computer utilization (MPCU)[10].

The motivation for the work described in this paper has been to understand the adoption and infusion of EPGs in companies with voluntary use. The core research question has therefore been: *What characterizes the use of an EPG among developers in a medium sized Software House?* In answering this question, the paper is organized as follows: the next section describes the research methodology and the research site that we call Company X, section 3 describes the findings that emerged from the grounded theory study, followed by discussion and conclusion in section 4.

2 Research Method

The research method followed in this study is that of *grounded theory* as described by Strauss and Corbin [12]. Grounded theory is a research method that seeks to develop theory that is grounded in data systematically gathered and analyzed. It is a form of field-study that systematically applies procedural steps to develop an exploration about a particular phenomenon. As Strauss and Corbin [12] explained: A grounded theory is one that is inductively derived from the study of the phenomenon it represents. Theory is derived from data, systematically gathered and analyzed through the research process.

In grounded theory, data collection, analysis, and eventually theory stand in close relationship to one another. A researcher does not begin a project with a preconceived theory in mind (unless his or here purpose is to elaborate and extend existing theory). Rather, the researchers begin with an area of study and allow the theory to emerge from the data. Theory derived from data is more likely to resemble the "reality" than if theory derived by putting together a series of concepts based on experience or solely through speculation (how one expects things to work). Grounded theories,

because they are drawn from data, are likely to offer insight, enhance understanding, and provide a meaningful guide to action. Although grounding concepts in data is the main feature of this method, creativity of researchers also is an essential ingredient.

2.1 Study Context

The context for this research is Company X, which is a medium-sized software company with approximately 150 employees in six organizational units. A separate group within Company X, called the SPI group, is responsible for building competence on software development processes, methodology and supporting tools. This group also has responsibility for coordinating the use of processes, methodologies and supporting tools across Company X projects. This responsibility includes the development, support and institutionalizing of the company's EPG. The EPG has been developed in close cooperation with the intended users through small workshops and meetings were the company's best working practice has been mapped. The EPG has been introduced to the employees through department meetings, and through training and direct support from the SPI group. The EPG is mainly supplementary to the company's work procedures based on ISO standards and, thus, of a voluntary nature.

The specific EPG examined in this research consist of custom-created process models for the company's internal use to provide process directions for the complete software life cycle. The EPG use fundamental concepts from the Capability Maturity Model (CMM), Microsoft Solution Framework (MSF) and Rational Unified Process (RUP). The model is meant to be scalable to both minor and larger assignments.

The EPG is web-based and available through the Intranet, reflecting the company's best working practices. The model defines a series of work phase definitions providing activity descriptions, flow charts, guidelines, procedures, standards, checklists, and templates for deliverables. Verification and validation requirements and criteria for approval of project deliverables are included in the definition of milestones and decision points.

The researchers were not involved in developing or introducing the EPG in Company X. The EPG was introduced two years before this study started.

2.2 Data Sources

Data were gathered through semi-structured interviews. We used the following questions as a starting point for the interviews: How often do you use/access the Process Guide? Do you like using tools like the Process Guide? Have you used tools like the Process Guide previously? If yes: purpose, frequency, how was it? What do you see as the main purpose with the Process Guide? Who do you think benefits mostly from using the Process Guide? What features of the Process Guide do you use most? What tasks do you use the Process Guide to help you with? Do you find using the Process Guide beneficial to your work? If yes, what were the benefits? Do you think there are disadvantages with the Process Guide? if yes, in what settings? What do you think should be improved about the Process Guide?

If the person interviewed did not use the Process Guide we added the following questions: Why do you not use the guide? Do you use any other resources to support your work?

We planned to interview 20 persons. Five department leaders was asked to point out two persons from their department they thought preferred to use the EPG, and two persons they thought did not use it or were against using it. One person could however not attend the interview. This selection was done to find out why people use the EPG, and why they don't use it. Because of this selection strategy, it is not possible to say anything about the average users of the EPG. The interviewer did not know which person belonged to which category. The interviews lasted 20-30 minutes, and have been recorded and transcribed in full for the analysis.

Eighteen of the subjects had software development as their primary job function, while one worked with administrating the databases. Four of the developers were also project leaders. Five of the subjects worked alone or together with another person on a project, eight persons worked in projects consisting of three to six persons, two persons worked on projects with six or more people, and four persons did not work in projects.

2.3 Data Analysis

In the data analysis we first used open coding followed by axial coding, and then selective coding [12]. Open coding is the analytic processes, through which concepts are identified and their properties and dimensions are discovered in data. Events, happenings, objects and actions/interactions that were found to be conceptually similar in nature or related in meaning, were grouped under more abstract concepts termed "categories". A category stands for a phenomenon, that is, a problem, an issue, or an event of a happening that is defined as being significant to respondents. The product of labeling and categorizing are the basic building blocks in grounded theory construction.

After the open coding, where we created concepts and categories; it was time to create connections between categories and its sub-categories. This is called axial coding. In this study we used a computer program called NVivo [13] for the qualitative data analyses. During the selective coding, where we integrated and refined the theory, the matrixes generated by NVivo were very important. We use quotes from the interviews in the presentation of the results.

3 Research Results

3.1 What Is the Intention of Introducing the EPG in the Company?

When asked about the intention of introducing the EPG in Company X, several interviewees said that the EPG would make it possible to deliver faster, work more efficient, and to lead the projects better. One said that *"It must be to deliver faster, better, and with better quality, than if you did not use it. You will produce projects that are similar to each other, and this makes it easier for new people to join the project and do a job"*. It was also commented that the EPG makes it easier to learn from former experience.

The interviewed subjects also meant that the EPG helps the developers carrying out their work, by describing how they can perform their job. It was also stated that these

descriptions help them work more similar. One said: "*Get better flow in the project, make us work more similar, and then you don't forget the things you need to do*". Another said "*You need it in big projects, where several people work together, and you need to disseminate knowledge, or should I say that we are able to understand each other"*.

Another commented that "*it makes the workday easier for us, you get good templates, and it is pretty obvious what you need to do in which phase. You use the checklists to make sure you don't forget things, and then you do it the right way.*"

Everyone stated useful aspects of the process guide independent of use level. The perceived usefulness is discussed later.

3.2 Who Needs the EPG Most?

The Project Leaders and Department Managers

It was a common opinion that the project leaders and department managers need the EPG the most. A developer said: "*I see the need for a process model, but what I really need as a developer is the help with using less time documenting what we do. The EPG helps you documenting the process, and this is very helpful for the project leaders*". Another said that "*The project leaders are sitting closest to the project model, hence the project leaders will use it most*".

Another comment was: "*It is very important for the project leader, but it is also important that the project leader communicate the EPG as a positive thing to the developers, and not as some garbage we have to use. This is important. But I think it is more useful for the project leaders, because they enter the EPG, use the checklists and verify that the phases are completed as planned.*"

The developers not using the EPG also meant that project leaders were the main target group of the EPG.

People Working Together

Several of the interviewed subjects meant that developers working in team benefit from the use of the EPG. One said *"People working together benefit from using it, because they need to work in the same way"*. Another said "*it is developed for the persons developing products, which means it is useful for everybody involved in the project. But I suppose it is more useful for the project leader."*

Developers

Some meant that the EPG is most useful for the developers. One said "*we (the developers) have most use of it, since we don't have to find all the documents ourselves."*

Customers

It was also commented that the customer would benefit from the EPG. One said "*It gives the customer an opportunity to understand how we work, and that is very useful. I have used the EPG in meetings, just to tell them how we work together on a project"*.

3.3 What Is Used in the EPG?

Use Level

Six persons claimed they did not use it at all, and only two persons answered that they used it several times a month. Four persons used it four times a year, but one of them used it daily in the previous project. Four persons used the guide once a month. The three last persons used an alternative model because this was required by the customer. The alternative model only consists of a few templates, and is therefore not used frequently. We were a bit surprised by the low use level of the guide.

Templates and Checklists

It was a common opinion that the templates and checklists were used most frequently. One said "*I download the template, and then I write the vision. I also use the checklists*". Another said: "*I have a few entries when I need the checklist for the Freeze process.*" And another said that "*I don't use the checklists, only the templates. It is especially in the beginning of the project, where I need the templates.*"

Also persons claiming that they did not use the EPG reported using the templates. One said "*I continually collect the templates from the guide"*. Another said: *"The only thing I know is that I'm supposed to write the three documents. I collect the templates from a place on the intranet*".

Other Functionality

In addition to templates and checklist, it was reported usage of coding standard, tips and tricks, good advices, and user documentation for the test system

Alternative Sources

The subjects with no or low use reported that they used alternative sources/techniques when they needed support. Talking to other developers was the most important method. One said: "*the coding standard is oral. People talk in the corridors, and there is a person that most people listen to, and he acts like the oracle.* Another said: "*… it is important that you know people. You must know whom to talk with if you are going to have a chance of knowing how to solve the tasks. …if you are new here, you are not helpless, but it takes a while before you are up and running.*

3.4 Perceived Usefulness

We wanted to explore how useful the EPG is perceived by the developers. *Perceived usefulness* is defined as the degree to which the subjects believe that using the EPG would enhance his or her job performance. In other words: How does the EPG affect the job performance, productivity and quality of work? Does the EPG influence how easy it is to do the job, and do the advantages of using the EPG outweigh the disadvantages?

Productivity

Some of the interviewed subjects claimed that the EPG decreases the productivity. One said "*I think it steals focus away from the tasks that are to be done. On the small projects that I work on it is better to have a good dialogue with the customer."*

Several developers argued that it decreases the productivity, since you have to produce a lot of documentation that is not really needed. One said "*You have to produce a lot of papers, and perhaps nobody reads them later*". Another said "*A lot is written in all the documents, and there is little information in connection to the project. I'm a typical realist, and I'm not fond of writing a lot. I like lists of keywords and checklists. I have seen a lot of documents with a lot of words and little content*".

One stated that the EPG decreases the productivity in the beginning of a project: *"In my opinion, the EPG is very useful. There are more advantages than disadvantages. But it is very hard to get the entire company to use the tool, because in the beginning you need to invest more time than you get back. You need to use it in the entire project before you understand the usefulness of the EPG."*

EPG Makes Work Easier

It was a common opinion that the EPG makes the work easier, since it supports a lot of useful checklists, and helps finding the right documents and filling them out. One said "*You have a template to follow….You don't need to think about what you should do*". Another said "*It gives you a very good start on the document. All the things you need are written there*". But there was an uncertainty about the value of the documentation produced. One stated "*It is useful for the developers, since we do not need to find all the documents ourselves. The question is: Who reads them? People mostly want Power Point presentations."*

Some stated that the EPG makes it easier to perform quality assurance in the project. One said *"As I mentioned earlier, this is quality assurance in a positive direction. It is much easier than big Quality Assurance documents that never get updated, and are only taking up place in your shelf"*.

The EPG also makes it easier to get an overview of the development process. One said *"it helps giving you a good project overview. Since all projects are performed the same way, it is easy for me to look into another project to find information I need"*. Another said *"It is very useful to have an overview of the different documents that need maintenance in connection with freeze and delivery. Then the model justifies it selves."*

Some stated that the EPG makes it easier to follow the process, but does not increase the productivity. One said "*I'm not a formal person. There are a lot of really good documents in the EPG, but there are also some documents that are NOT very good. The good documents are those describing the process, but the developers need help documenting what they do. We need help to write system documentation."*

Quality of Work

The EPG was also perceived to increase the quality of work, since it gives instructions on what you need to remember. One argued that "*I see it as a useful tool. You have a customer, and then it is important not to make a blunder. It is OK to have the templates you can access, to clarify scope, risk and such things, and that you agree with the customer what you are intended to produce*".

3.5 Perceived Compatibility

Perceived compatibility is defined by Rogers [8] as the degree to which an innovation is perceived as being consistent with the existing values, needs, and past experience of potential adopters.

Compatibility with Project of Different Size

The majority of the interviewed persons had the opinion that the EPG was not suited for smaller projects. They thought the use of EPG would cost them too much, and that it draws focus away from central activities in the project. One said: "*in small projects, I'm squeezed on how many hours I can use… …It takes focus away from writing good code.* One other said: "*We have the big documents including everything, but we also need the smaller documents telling us what is required as a minimum when you have a small project"*.

It was the opinion among several of the interviewed persons that the EPG supported bigger projects. One said "*it is built for bigger projects. With bigger projects I don't need to have a lot of people working on it, but it need to be a team working together for a specific period of time"*.

Compatibility with the Early Phases

It was the common opinion that the EPG is compatible with the early phases (before programming starts). The most intensive use of the EPG is in this phase. One said "*you start with conceptual design and GUI prototyping, and then you go over to the implementing phase, where you only write code, then you don't use it any more."*

Compatibility with the Implementation and Maintenance

Several persons did not think that the EPG supports the implementation process. One said: "*After we start programming, we don't use the EPG at all*". Another said: "*The developers need help documenting what we do. It's not good at support system documentation."*

Another said that "*as I earlier told you my opinion is that the implementing phase has very little support. I think the ambition for the process model is very good since the technology changes very quickly. It will not be easy to maintain templates and documents describing the technology, because it will be outdated very soon."*

When it comes to maintenance some meant that the EPG did not support this phase, and others meant it supported this phase good.

Compatibility with the Need for Documenting

The EPG only supports the manual part of the documentation process. It's a strong wish from the interviewed subjects that the EPG should automate the documentation process. One said: "*I was a bit disappointed in the beginning. I didn't see it as a tool, only as a pile of templates located in a fixed place. The glue is missing …. If you work with different models in a project you need to cut and paste to update them in the different word documents.*"

It was also commented that "*you need to write a lot of documents that may never be read. You need a living design document that describes what should be in the*

product”. Another said “*Of course we had used it more actively, if it had given something back, if it had helped us besides being a guide that describes how we should work.”*

Possibility to Adapt the EPG to the Project
The majority of the persons meant that it was necessary to adjust the EPG to the activities they performed. One said: “*You need to do some work with the templates. The development projects vary a lot here, if you are going to write a software component with an interface and a GUI, compared with when you are supposed to write something that includes both hardware and software. Most of the templates are made for software development.*”

Another said: *“If you are going to follow everything that is written there, sooner or later you will get problems. There must be an opening to adapt to the task you are performing. Therefore it is important that the EPG is more general”*.

3.6 Perceived Ease of Use

Perceived ease of use refers to the degree to which a person believes that using a particular system would be free of effort [3].

Several of the interviewed subjects stated that the EPG was difficult to learn. One said “*First time I used the EPG I used plenty of time to get into it and to understand the systematic*.”

Some claimed that it was easy to use and some claimed that it was a bit difficult. One said *“I think it is easy to use, you see a graphical representation of the whole process, you just go were you need to go and click, and then you get to the document you need.”* Another said *“If you are using a template, you need to read a lot before you can start writing. You open it and start filling out the fields on the first page. Author, title, and then you need to fill inn a lot of field codes. You have to do a lot of manually operations. And then you think: Can't they automate these operations if they want this kind of documents. It is very irritation, can't they fix this. It's even more irritating when you are working inside the document, and you still have to do a lot of manual operations.”*

Some commented that it was cumbersome to use. One said *“It is too much paper and they are to general, and you use a lot of time to understand the meaning of this schema.”*

3.7 Subjective Norm

Subjective norm is the degree to which software developers think that others who are important to them think they should use the EPG. This suggests that perceived social pressure to perform the behavior will influence a person’s intentions [1]

One of the department leaders preferred not to use the EPG. One said *“It is decided on the top level that we must have a process model, but my boss does not have the same opinion. And since he think the guide is heavy and bureaucratic we don't use it much”*.

Some claimed that it was very important that someone really wants and needs the produced documentation. One said *"if they want people to use it someone have to pressure our leaders, to get them to ask for the documentation we make. As long as they don't ask for the documents we are supposed to create, there will be a low use level.*

One said that "*on external projects where I have been the only developer, and where the project leader does not ask for it, we don't use it.*"

4 Discussion and Conclusion

A qualitative field study was performed to investigate *what characterizes the use of an EPG among developers in a medium sized Software House*? Each department leader pointed out two persons they thought preferred to use the EPG, and two persons they thought did not use it or were against using it. Only half of the interviewed subjects worked on medium sized or big projects. This was a low number consider that the main activity in Company X is project work. The reason for the low number could be the selection strategy.

It was a common opinion that the project leaders and managers are the user groups that benefit most from using the guide. None of the developers in the "no use" group see themselves as the primary target group for the guide, but several of the developers using the guide meant that they were the target group for the EPG. This could mean that some of the potential users don't use the EPG because they do not think the guide is made for them.

According to TAM, systems that are perceived to be easier to use and less complex have a higher likelihood of being accepted and used by potential users. The EPG is perceived as both useful and not useful. The checklist and templates are commented as the most useful functionality of the EPG. Several of the interviewed subjects that perceive the EPG as less useful, commented that they are missing integration with other tools. Especially help with documenting and updating documents. We have done a similar study' in a company called Firm [6], and here we found one of the key success factors for the EPG to be integration with other tools, e.g. a system for process tracking.

Several had the opinion that that the EPG was not compatible with the implementation process, and therefore they did not need it for this phase of the project. One important observation was that none seemed to miss support from the EPG in this phase, and therefore it is possible to conclude that the developers have the support they need. Several of the subjects interviewed did not participate in the coding phase of the projects, and this of course influence the use level of the guide. When analyzing the EPG's compatibility with the different project types, it was found that several thought the guide was primarily for bigger projects, and not for small projects or people not working on projects. The majority of the people with no or little use did not work on any project, or worked on really small projects. This could also be an explanation for why some of the developers did not use the guide. On the other hand several stated that it was easy to adjust the templates to the different project types.

Several persons reporting little use or no use at all said that they used several templates. They didn't consider themselves as users, even though they were active users. Maybe they did not have a clear understanding of what was meant with the EPG?

One important finding that influences how the EPG is perceived and used, was that one department leader did not fancy the guide, and that several felt that the leaders really don't read the documents or ask for the documents produced.

In [6] we found that it was important to involve the users strongly in developing the EPG to succeed with institutionalizing. This was not the case of the persons interviewed in Company X. Only two persons reported such involvement, and this could explain the low use level.

We have found that implementing an EPG is not a straight forward process. To succeed with the implementing process it is important to consider:

- Use time and effort to advertise how usefulness the EPG will be for the target users
- Templates and checklists are seen as very useful
- Automation of the documentation process, and integration with other tools
- In what degree support during implementation is necessary
- Implementing functionality for tailoring the process to different projects types or activities, if the whole development organization is going to use the EPG
- Anchor the EPG among the leaders and involve the users in the EPG creation process

However, further research is needed to examine EPG usage in a wide variety of organizational settings. It will be important to study documentation from the projects were the interviewed subjects has worked to find out more about how much the EPG is used in the organization. It will be interesting to find the use level of those that do not think they use the guide, but reported use of several templates.

Acknowledgments

This work was supported by the Research Council of Norway under Grant 156701/220. The authors wish to thank all the subjects of Company X that took part in this investigation.

References

1. Ajzen, I. (1991) The Theory of Planned Behavior, *Organizational Behavior and Human Decision Processes*, vol. 50, pp. 179-211.
2. Baruch, Y. (1999) Response Rate in Academic Studies - A Comparative Analysis, *Human Relations*, vol. 52, no. 4, pp. 421-438.
3. Davis, F. (1989) Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology, *MIS Quarterly*, vol. 13, no. 3, pp. 318-339.
4. Fichman, R.G. and Kemmerer, C.F. (1993) Adoption of Software Engineering Process Innovations, *Sloan Management Review*, vol. 34, no. 2, pp. 7-22.
5. Kellner, M.I., Becker-Kornstaedt, U., Riddle, W.E., Tomal, J., and Verlage M. (1998) Process Guides: Effective Guidance for Process Participants, *Proceedings of the Fifth International Conference on the Software Process: Computer Supported Organizational Work*, Lisle, Illinois, USA, 14-17 June, pp. 11–25.
6. Moe, N.B., Dingsøyr, T., Dybå, T., and Johansen, T. (2002) Process Guides as Software Process Improvement in a Small Company, *Proceedings of the European Software Process Improvement Conference (EuroSPI'2002)*, Nürnberg, Germany, 18-20 Sep.

7. Moore, G.C. and Benbasat, I. (1991) Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation, *Information Systems Research*, vol. 2, no. 3, pp. 192-222.
8. Rogers, E.M. (1995) *Diffusion of Innovations*, Fourth Edition, New York: The Free Press.
9. Scott, L., Carvalhoa, L., Jeffery, R., D'Ambra, J., and Becker-Kornstaedt, U. (2002) Understanding the Use of an Electronic Process Guide, *Information and Software Technology*, vol. 44, pp 601-616.
10. Thompson, R., Higgins, C., and Howell, J. (1991) Personal Computing: Toward a Conceptual Model of Utilization, *MIS Quarterly*, vol. 15, no. 1, pp. 125-143.
11. Venkatesh, V. and Davis, F. (2000) A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies, Management Science, vol. 46, no. 2, pp. 186-204.
12. Strauss, A., and J. Corbin. 1998. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*., Sage Publications.
13. QSR International Pty Ltd (2002). www.qsrinternational.com

Knowledge Mapping: A Technique for Identifying Knowledge Flows in Software Organisations

Bo Hansen Hansen and Karlheinz Kautz

Department of Informatics, Copenhagen Business School
Howitzvej 60, DK-2000 Frederiksberg, Denmark
{bo,karl.kautz}@cbs.dk

Abstract. Knowledge is a key parameter for software companies' survival, as the ability to continuously become better at producing services relies on the organisation's abilities to develop and utilise the intellectual competencies, which its employees possess. These abilities depend highly on the organisation's capability to share knowledge and thus on the way knowledge flows in the organisation. In this paper we present a knowledge management perspective on software process improvement, and describe a technique for mapping the organisational knowledge flows in a software company. The results show that the technique successfully helps the organisation to select relevant focus areas for planning future improvement initiatives. The study further explains four distinct critical situations, which can be identified in a knowledge map; Hubs, Black Holes, Springs, and Missing Links. Each covers potential problems in the organisational flows and therefore can provide guidance for organisational process improvements.

1 Introduction

Knowledge management is a key concern for organisations because knowledge is the primary asset in the post-industrial era [21]. The production environment and infrastructure play a diminishing role and knowledge, intellectual capital, and the management of this knowledge, a growing one. This is also true for software companies: their main asset does not consist of the technical development environments, tools and platforms, but of the ability to develop and utilise the intellectual competencies their employees have to create services of value for their customers. Software process improvement (SPI) is concerned with strengthening the software development abilities, which means that a primary task for SPI research is to strengthen the knowledge managing abilities for software developing companies.

In this paper we present a knowledge management perspective on the software process improvement and provide a means for identifying and analysing the knowledge flows in an organisation. We do so by developing a technique for mapping the knowledge flows between the organisational actors, and show how this technique is applied within an organisation, and how it is used to analyse the organisation's knowledge sharing capabilities. The technique is used to point to areas where improvements are possible, and to locate ideas for how to make these improvements happen.

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 126–137, 2004.

The paper is structured as follows: In the following section the theoretical background for the mapping technique is explained. Section 3 describes the research method, and section 4 provides a short description of the context, in which the study is conducted. Section 5 gives a detailed description of the knowledge mapping technique. Section 6 describes the results of applying the knowledge mapping technique in the organisation, and these results are discussed in section 7. The paper is concluded in section 8.

2 Background

In the last decade the primary approach for strengthening the knowledge management abilities has been by leveraging the organisational knowledge with an outset in previous experiences [1, 12, 15, 17]. In line with these more general knowledge management methodologies, most SPI approaches base the improvement effort upon previous experiences [11, 22]. Different approaches to SPI focus on different origins for this experience; either by concentrating on learning from the organisation's own experience like the software factory approach [4] or by taking as its starting point a model based on the empirical findings from the software industry at large like the Capability Maturity Model [20] which provides a stepwise roadmap to improvement based on a formal assessment of the organisation's development capabilities. The general idea is that by analysing the past it is possible to become better in planning for the future by learning from experiences. This idea builds upon an assumption about that even in a dynamic environment many factors are constant from project to project, and by carrying out incremental improvements based on evaluations and analysis of previous development, it is possible to improve future practises. A basic requirement for experience based learning is that the knowledge-about-practice is shared among the organisational members otherwise only those who were involved in a particular project gain from the experience. Thus capabilities for sharing knowledge are an important issue in SPI.

3 Research Method

This paper reports on a study of a software company in Denmark. The overall study is based on a close collaboration between the organisation and researchers and is inspired by Mathiassen's "Collaborative Practice Research" (CPR) approach [18]. This approach is based on a three step repeatable process with different research activities addressing different knowledge types. The understanding of the subject area is achieved through collection and interpretation of data about practice. On this basis normative propositions to support practice can be designed, and through intervention these propositions can improve practices. The outcome of the effort can then be used as a basis for a new understanding and triggers further improvement cycles.

The study covers 10 months of work and the collaboration is arranged around a steering committee in which two representatives from the organisation and six researchers participate. The committee, which meets approximately bimonthly, func-

tions as a planning forum, where new initiatives are planned and the results hereof are discussed.

Several studies [3, 5, 9, 10, 16] have shown that actual practice, even in organizations with strong formal structures, often differs from what is described in formal plans and guides. This implicates that to understand how knowledge flows it is necessary to conduct an examination of the organizational practise as it is not enough to rely on formal descriptions and organisational charts and plans.

Knowledge flows can take various shape including forms that the organisation's members not explicitly are aware of, which means that they are not easily observable. Therefore to analyse the situation and to assist the planning and design of actual interventions, we use a number of qualitative data gathering techniques that go beyond formal assessment approaches, which only turn the attention to process problem areas defined in their underlying model and do not support alternative views, like a knowledge management perspective, and the identification of difficulties outside their scope [13].

One researcher is present in organization once a week, which has provided the research with detailed insight into many "taken-for-granted" situations. He has also participated, both with active and in observing roles in a number of meetings and employee courses. The researcher's ability to ask "What happened just there?", allows to explore non-formal communications and practices, and gives a broader view providing a more detailed understanding of the organisation. The information gathered from the visits is documented in a diary and an encyclopaedia including organisational constructs. The researchers have conducted eight semi-structured interviews in three interview rounds with employees from all organizational levels. These interviews are based on an interview guide and are used to learn about the respondents' interaction with other organisational members and units. They also focus on how the different employees coordinate and organise a working day, and what tools and other means of communication carry their interactions. The interviews were tape-recorded and a resume was written afterwards. They were verified through discussions in the steering group as part of the mapping process itself (see section 5.4).

Finally, the study included an analysis of artefacts in connection with the software development processes, which in this context are templates for reports, manuals describing organisational processes, computer based tools used in the development projects.

4 Context: The Case Company

The case organisation employs more than 300 people, mostly system engineers with an academic background. All development is organised in projects and the project teams are quite stable: most developers only work on one project at a time. The projects are large, 55% of them larger than 10.000 working hours. Continuously 20-25 customer projects exist simultaneously and the typical project develops new applications for one customer, but some of the applications are off-the-shelves products with long lasting projects for maintenance and further development.

The company is highly involved in developing their abilities to produce software, and has 8 years ago adopted the CMM as their basis for conducting SPI efforts. At the time of the intervention the company was at CMM level 3, but has now reached level 4. The efforts are rooted in a SPI-team, which employs about 15 full time staff. The improvement effort is mainly based upon the development and implementation of a Business Manual (BM), which is an Intranet based manual describing how project management and software development should be carried out. The SPI-team participates in the education of the projects members in how to use the BM. The individual projects have to comply with the BM, and are audited from time to time by an internal auditor.

The company's representatives in the research project's steering committee are the manager of the SPI-team, and the manager for the business area, in which the SPI-team is located. The company management recognises the need for investing in initiatives for continuous organisational development from which it expects high returns on investments. The company has implemented an extended measurement program to establish 'quantitative management'. This is integrated with a balanced score card approach and documented in a yearly published "Intellectual Capital Report" audited by a large consulting company. The organisation has been awarded with "Entrepreneur of the Year 2003" medal in Denmark and is recognised as a competent player in the Danish market. Applying the learning organisation paradigm the company is implementing a knowledge network initiative, which is based on formal professional networks with the purpose to strengthen the abilities to learn and to "incubate" new improvement initiatives.

The study takes it outset in a situation where members of the SPI-team have experienced occasions where knowledge does not flow as desired. In this context the SPI team and the researchers wanted to perform a thorough investigation of in what condition the organisation's knowledge sharing capabilities are, to examine in which areas improvements are possible, and to locate ideas for how to make these improvements.

5 Knowledge Maps

To perform the desired analysis, we developed technique to map the organisation's knowledge flows. This technique has been inspired by existing techniques for addressing complex problem situations, namely rich picture drawing and problem mapping.

5.1 Rich Pictures

A known technique for understanding complex problem situations is Rich Picture drawing [6, 19] where rich pictures are defined as "the expression of a problem situation compiled by an investigator, often by examining elements of structure, elements of process, and the situation climate". These pictures seek to outline a holistic drawing of the problem area instead of focusing on specific problem situations. The build-

ing blocks of a rich picture consist, besides the structural and process elements, of the situation climate. The rich picture technique requires a thorough data collection e.g. based on interviews with representatives of all different actors. A rich picture contains different viewpoints, potential disagreements or conflicts allowing for multiple perspectives at one time. The technique does not favour one way to actually drawing over another, but leaves this to the picturemaker(s), but the basic elements have to be visualised. The visualisation is an important feature as hereby the picture can be used as a means of communication. The technique enables an outsider to draw a complex human activity system in a holistic picture. People can easily comment on a picture, maybe they do not understand it the same way as the maker or as it was intended, but it suits as a conversation enabler or discussion starter, which may lead to a common understanding of the problem situation.

5.2 Mapping Techniques

Various mapping techniques have been introduced to analyse problem areas, which are not fully or differently understood by different stakeholders. Maps are defined as being "an interpretive description of a situation" [14] and are as such an interpreted model of reality i.e. a map consists of selections of relevant details of the mapped situation and provides information about what the mapmakers find relevant. Maps provide a possibility to gain an understanding of a complex problem situation and at the same time facilitate a common understanding among different stakeholders. Different mapping techniques can be used in e.g. systems development projects to collect and organize relevant knowledge [14]:

A *diagnostic* map consists of a presentation of the results of a root-cause analysis in which the mapmakers discuss experienced problems and seek causes and effects to find alternative approaches to avoid the problems. *Ecological* maps outline the connections between problems and the organisational context of the problem. The mapping technique seeks to define which conditions influence the problem situation and whether these are to be found within the project team or outside. *Virtual* maps outline desirable future situations, and take their outset in asking "What do we want?" and "How do we get there?". Further different alternatives and their outcomes are considered, and the result is a description of different routes to achieve the desired outcomes. *Historical* maps have a retrospective perspective in how they map the past: a previous project is described with respect to its key events, and relates these to the actions and conditions. Historical maps can be used to learn what might be critical factors in a similar future project.

In our study diagnostic maps are especially of interest as they are used to understand the current situation of the project or organisation and thereby provide an overview of the status. The mapping technique incorporates participant involvement as a necessary condition to have any relevance or validity. The technique, besides finding alternative approaches, also has a therapeutic effect, as the involved have to agree upon, which problems are actual problems and how they affect them in their project roles. Mutual understanding and knowledge sharing are elements in the technique,

which makes it an integrated part of the problem solving method. The maps are not the only outcome of the technique, the process of mapmaking is an important part of the technique as well. The maps show important features of the organisation and the analysis of the various flows and grouping of flows provide the input needed for further action.

5.3 A Knowledge Map

For analysing the knowledge flows we utilise the strengths from each of the underlying techniques. A knowledge map consists of the elements of structure, elements of process, and a representation of the climate within these two exists, each of which will be described below.

The basic elements of a knowledge map are the different actors involved in the mapped situation. This comprises the formal organisational constructs like the organisational units, project teams, individuals, etc. But also important artefacts regarding the flows have to be considered. This could be reports created by some to be read by others or software tools like an error reporting system. These elements constitute the basic nodes of the map. The actual choice of drawing symbols is not important, it is important to choose representations, which are understandable, and direct the viewer's thoughts to the relevant elements of the experienced reality.

Between the structural elements the knowledge flows represented by lines appear. A flow comprises interaction between various structural elements, and can consist of informal discussions as well as strictly formal half-year reports; what is important is that some actor acknowledges it as means of knowledge exchange. Some flows are bi-directional, and some unidirectional, and some might be both depending on who defines them! The flows can differ with respect to their frequency and to the amount of information they contain, both important features to provide an understanding of the overall flow between various elements. It is useful to model what type of information is contained in a flow. The importance ascribed to flows is also a significant feature. A potential misalignment might show up, when climate of the interaction are described which is the topic of the last element of the knowledge map.

The climate is a key information provider. It contains the expressions about the circumstances under which interaction takes place. This contextual information is a major indicator for pointing out problems, and contains multiple perspectives depending on the viewpoints brought forward. It can consist of the actor's thoughts about why a situation is experienced as good or bad, and thoughts of how this situation could be improved, it can be the actor's expressions of where conflicts arise, or it can be other comments about the knowledge flows.

5.4 Knowledge Mapping

Our approach to using the technique consisted of two phases, a preparation phase conducted by a mapmaker, and a collective mapping phase in which the steering committee together created the actual map.

The preparation phase was conducted by one of the researchers, the mapmaker, who had conducted the data collection. He created a preliminary knowledge map of the organisation. The primary reason for initial mapping was to let him prepare himself before the actual mapping. This generated map could be followed during the mapping phase, and kept the task on track, even when discussions moved in different directions. The map prepared in advance secured that the, for the original mapmaker, relevant aspects were covered. It allowed the mapmaker prepare a note of questions on topics, which he felt were not explained satisfactory in the collected data. Drawing the map in advance also provided the opportunity to list what he saw as major problems and improvement areas. These were used as discussion topics, in situations where the process needed stimulation. The secondary reason for letting him create a map in advance was that it was done without interference from others members which enabled him to record his understanding, and thus provided the whole mapping phase with the quality of having an outsider looking at the organisational system; this external input would not have been so clear, if organisational representatives would have had the opportunity to align the mapmaker's views along as they were presented.

The actual mapping phase, the joint creation of the map, was conducted in the steering committee. It consisted of four steps and took place in a day long meeting. The steps together functioned as verification, clarification, and extension of the mapmaker's preliminary map. The various elements of the map were discussed in an open atmosphere. The mapmaker was acting as the meeting leader and he introduced and facilitated the process. He was responsible for leading the discussion, and for documenting the results on a white board. The 1st step consisted of drawing all important elements of structure on the white board. The mapmaker selected one area of the organisation to begin with and started drawing organisational units, artefacts, people etc. listed on the preliminary map. While doing this, he presented his understanding of the role of each of these. This promptly initiated a discussion among the participants, because some of the descriptions were not correct accordingly to their opinions.

To represent individuals we used "stick figures", some with additional characteristics because they represented individuals with specific roles and importance. We used groups of stick figures to represent certain organisational units, like development projects and sometimes we encircled these to mark specific boundaries with the context. We used a picture of a document to represent written reports, and their formal abbreviations to distinguish them. Furthermore we used different symbols for technical systems; these symbols were easy to understand for all participants, and were if necessary, they were equipped with explaining text. The 2nd step consisted of describing all different knowledge flows. The meeting leader introduced a specific flow between two or more people or artefacts, and described his understanding of it. This quickly facilitated a discussion outlining special cases and corrections, providing clarification and richness to the map. This step also introduced overseen people, roles, and artefacts to map. In the 3rd step the climate resp. context was added by providing statements identified during the data analysis. The step included a discussion of which flows were found problematic, and which flows were missing. When we conducted this step a lot of new problematic issues surfaced, and the organisation's rep-

resentatives became kind of defensive. The meeting leader directed the dialogue away from this defensive position, and emphasised the opportunity to develop ideas for improvements. The focus instead was directed towards addressing the question why some members of the organisation experienced these problems, even if the SPI-team's representatives did not acknowledge them. Here the mapping technique showed its value, since it was possible to show that some might experience problems or conflicts, where others were not seeing them. We used a new colour to highlight the problematic areas, and denoted them with a large exclamation mark. We also used this step to indicate on the map where new ideas and initiatives originated by marking these with a light bulb. The 4th step consisted of analysing the identified problems to understand their roots and causes. This discussion supplemented the map with a list of identified improvement areas. The map allowed the diagnosis of each problem with its particular context with respect to structure and process, which made it easier to identify what parts of the organisation were affected, and maybe should be involved in the search for a solution. A primary result from the analysis pointed out, that key information regarding development projects was collected in formal project reports. The project managers responsible for creating these reports felt that these reports never reached the SPI-team. As a result the project managers spent less time providing this information, thus making fewer experiences available. On the other hand the SPI-team felt that the reports offered very little relevant information, and therefore did not spent much time analysing them. This example of a negative cycle, which, if not taken care of, eventually would stop a means of knowledge flow, shows the general idea and potential benefits from the mapping technique.

Even though the steps above are described as a linear process, the actual mapping was characterised by letting the discussion follow interesting topics, and thereby mapping larger organisational "chunks" rather than finalising each step at a time. Thoughts were allowed to drift, and the discussion moved more iteratively from one topic to another and the meeting leader had his own map to fall back on, when the process needed to proceed. To allow for later analysis, the last item on the agenda was the documentation of the result, which consisted of photographing the white board.

6 Result

Beyond the above-mentioned example, an important result of our analysis, which confirms the map's function as a device to support understanding and consensus, was the broader comprehension of the knowledge flows by and among the participants of the session. When the participants were presented with other viewpoints than their own, they had to reflect upon these, and this had an impact upon how they saw the situation, and provided them with a better basis for an analysis in the future.

The use of the mapping technique has also led to the development and implementation of a new project evaluation concept, which integrates the diffusion of experiences through the formal knowledge networks. Beyond this, we recognised four clear categories of characteristic situations, which even in our complex map were rather distinct.

A specific individual or organisational unit with a large number of connecting knowledge flows, we call a *hub*. In our case i.e. one person had several roles in various parts of the organisation, which appeared in the map as many flows leading to him. A hub might be a useful to have in an organisation, if it can cope with the knowledge flowing to and from it, and can effectively use this information, but too many flows ending in one place might easily create congestion and thus a hub might end developing into a bottle neck slowing (or discharging) information, and thus become a hindrance for the organisation's ability to share knowledge

Black holes are places where no flows origin. This means that knowledge only flows one way towards this area. This might not be problematic, but, learning from experience is an important part of the organisational development, and when specific parts of the organisation are not feeding experiences back, it is not possible for other parts to learn. An example from our study showed that one of the organisation's knowledge sharing tools was not used by the employees making it a "write only" asset from which no knowledge were fed back to the organisation.

Areas from where lots of flows origin, but none are going in are *springs*. These might indicate potential innovative centres where lots of ideas are created and exported to the rest of the organisation. A spring might also point to an area, which is not using others' experiences. This is not necessarily a problematic situation, as it can represent a highly specialised unit, which does not need any input, but it might as well point to a potential problem. In our case the top management constituted a spring by continuously feeding the organisation with new ideas and suggesting new initiatives.

Missing links describe situations where a link would be beneficial, but for some reason is not there or not functioning satisfactory. As such missing links are more problematic to spot in a map. Similar organisational units might achieve advantages from having a close dialogue and sharing of their experiences, if no knowledge flow exists between them, a potential benefit might be lost. In our study many projects found it difficult to feed back their experiences to the SPI-team, which then had fewer experiences to base their work upon.

7 Discussion

In the case setting the knowledge mapping technique provided useful results, but there is no guarantee for success. An important factor, which supported the technique, was that the mapmaker and the other researchers had been present in the organisation for a long time. The members of the organisation knew the researchers well and trusted them. The trust is important when comes to the discussion of confidential information and an open-heartedly atmosphere is one crucial pre-requisite for a successful outcome [18]. If the participants feel that they cannot address all aspects of the organisation, and are afraid that some issues are not to be discussed, then the map and the mapping process will be limited and critical areas might not be uncovered.

Besides this "open-dialogue" effect, the researchers' long presence meant that the discussion went well. Having cooperated with employees and participated in meet-

ings and informal activities, the mapmaker and the other researchers knew the culture, overall values, the language, the artefacts and the tools used in the organisation. During the mapping process the organisational members did not have to explain all organisational concepts, and thus the conversation flowed easily and focused. The mapmaker could use the basic understanding as it enabled him to see the larger picture, and therefore made him better in combining various elements in (or not in!) the map. There is however a danger of working in the wrong belief of having a good understanding of the organisation. If the understanding is wrong, the misconception can lead to misunderstandings.

Our approach, which relies on and addresses all relevant personal to get a view of the practice as perceived by all stakeholders, a view which might differ from the official description, might be criticized as only SPI management and no actual project participants were present at the mapping session. Only secondary project and management information were directly available and there was a danger of getting a distorted picture. One might argue that involving more personnel in the mapping session and providing more resources for the process could avoid this. This however might increase the complexity of the process where an even richer picture might lead to problems of identifying the relevant issues. In our case however, resources were limited and the SPI-team did not want to involve more employees in the mapping sessions and the research team decided that for a pilot test of the technique a smaller forum would be more adequate. Thus, the conflicting demands were balanced and compensated by using a mapmaker, who was well acquainted with and trusted in the organisation and a process with an outcome, which was not restricted by management views and limited input was achieved. The resulting improvement proposals were beyond the participants of the mapping sessions, welcomed by other members of the organisation.

In this context the concepts of springs, hubs, black holes, and missing links have been helpful to structure the (discussion of) knowledge flows. The learning from experiences approach is a knowledge exploiting approach [8], which helps the organisation become expert in its field, but at the same time includes the danger becoming "skilfully incompetent" [2]. Springs can be a means of supplementing exploitation with explorative approaches. Another important aspect is the ability to identify where knowledge does not flow as intended, bottle necks caused by hubs, which can be a hindrance both for diffusion of new ideas, but also for feed back on effects of innovations. The same is true for black holes and missing links, which "point" per se to problematic areas.

The use of rich pictures also has to be considered carefully. Pictures can be easily handled by some whereas others have problems drawing and discussing via pictures [7]. Therefore it is important to facilitate the use of pictures by letting a meeting leader do the drawing the first time the technique is used. In our case the researchers, who all had experience with rich pictures, were present and could help the other participants in the use of the technique.

During the actual session, we also noticed the following items, which in the future might help to enhance and make the process simpler. A good idea is to use different colours for the different elements on the white board, as it later is easier to distinguish

between structure, flows and text. Another advice is to use a very large white board as the map can to become very rich, and thus includes very much detail and data – and it is much more difficult to embrace a white board very crowded with drawings at one glance. Leaving space between the different drawings makes it easier to avoid moving parts of the drawing, which can be quite problematic, when the process is well underway – it is difficult to make certain that all flows are moved along, and it can be very difficult to re-connect a flow later, if nobody remembers exactly what this flow represented. We did not provide the flows with a description of what they represented, but this might be a good idea, if some flows prove to be more interesting, and thus become the subject for a more thorough examination. By labelling the flows it will also be possible to verify that the same flows are not represented twice in the map.

Finally, it turned out to be helpful to start with the roles and units of the members present at the mapping session, they, as expected, certainly had something to add or correct, and this facilitated their involvement into the discussion.

8 Conclusion

Based on the insight that understanding the knowledge flows in an organisation will support software process improvement activities we describe in this paper how we in a close collaboration with the employees of a software company identified possibilities for improvements. For this purpose, we developed a knowledge mapping technique based on using rich pictures as a tool, visualising the relevant organisational units, individuals and artefacts, and their mutual communication streams or knowledge flows, including the context surrounding these flows. We used a range of qualitative data gathering methods throughout a ten months period to inform the application of the mapping technique in the organisation. We conclude that this mapping technique provided a helpful means to understand the complexity of a software company with many simultaneous projects. The technique produced valuable results for the organisation as it provided information about where and which improvements could be relevant.

In the specific setting, where trust and mutual respect prevailed, the technique proved useful and provided the organisation with valuable feed back, but the same setting might not lead to the same success in another organisation or with other participants. The technique as such is no silver bullet for creating improvement initiative candidates in every context. Our description and analysis however contributes to the existing body of knowledge in the field of knowledge-based software process improvement with a new promising support tool and can contribute as such to practical improvement work. It should lead to further research testing the mapping technique in different forms and within other contexts.

References

1. Andersen, I.: Choice of Organisation Sociological Methods - A Combination Perspective (in Danish). Samfundslitteratur, Copenhagen, Denmark (1990)

2. Argyris, C.: Knowledge for action - a guide to overcoming barriers to organizational change. Jossey-Bass, San Fransico, USA (1993)
3. Bansler, J. and Bødker, K.: A Reappraisal of Structured Analysis: Design in an Organizational Context. ACM Transactions on Information Systems 11, 2 (1993) 165-193
4. Basili, V., Caldiera, G. and Rombach, H. D. The Experience Factory. Encyclopedia of Software Engineering. J. J. Marciniak. West Sussex, UK, John Wiley & Sons, Inc (1994)
5. Brown, J. S. and Duguid, P.: Organizational Learning and Communities-Of-Practice: Toward a Unified View of Working, Learning, and Innovating. Organization Science 2, 1 (1991) 40
6. Checkland, P.: Systems Thinking, Systems Practice. John Wiley and Sons, West Sussex, UK (1999)
7. Checkland, P.: Soft Systems Methodology: A thirty year retrospective. Systems Research and Behavioral Science 17, S1 (2000) S11-58
8. Fitzgerald, B.: An Emperically-Grounded Framework for the Information Systems Development Process. Proceedings of International Conference on Information Systems (ICIS), Helsinki, Finland (1998)
9. Fitzgerald, B.: An Empirical Investigation into the Adoption of Systems Development Methodologies. Information and Management 34, 6 (1998) 317-328
10. Hansen, B. H., Jacobsen, D. and Kautz, K.: Information Systems Development Methodologies in Practice. Proceedings of International Conference on Information Systems Development (ISD), Melbourne, Australia (2003)
11. Hansen, B. H., Rose, J. and Tjørnehøj, G.: Prescription, Description, Reflection: The Shape of the Software Process Improvement Field. Proceedings of UK Academy of Information Systems (UKAIS) Conference, Glasgow, Scotland (2004)
12. Holmqvist, M.: A dynamic model of intra- and interorganizational learning. Organization Studies 24, 1 (2003) 95-123
13. Iversen, J. H., Nielsen, P. A. and Nørbjerg, J.: Situated Assesment of Problems in Software Development. The Data Base for Advances in Information Systems 30, 2 (1999) 66-81 (15)
14. Lanzara, G. F. and Mathiassen, L.: Mapping Situations Within a System Development Project. Information & Management 8, 1 (1985) 3-20
15. Levinthal, D. and March, J. G.: The Myopia of Learning. Strategic Management Journal 14, Winter Special Issue: Organizations, Decision Making and Strategy (1993) 95-112
16. Madsen, S. and Kautz, K.: Applying System Development Methods in Practice - The RUP example. Proceedings of International Conference on Information Systems Development (ISD), Riga, Latvia (2002)
17. March, J. G.: Exploration and Exploitation in Organizational Learning. Organization Science 2, 1; Special Issue: Organizational Learning: Papers in Honor of (and by) James G. March (1991) 71-87
18. Mathiassen, L.: Collaborative practice research. Information Technology & People 15, 4 (2002) 321-345
19. Monk, A. and Howard, S.: Methods & Tools: The Rich Picture: A Tool for Reasoning About Work Context. Interactions (ACM) 5, 2 (1998) 21-30
20. Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C. V.: Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-024). Pittsburgh, USA (1993)
21. Quinn, J. B., Anderson, P. and Finkelstein, S.: Managing Professional Intellect: Making the Most of the Best. Harvard Business Review 74, 2 (1996) 71-80
22. Aaen, I., Arent, J., Mathiassen, L. and Ngwenyama, O.: A Conceptual MAP of Software Process Improvement. Scandinavian Journal of Information Systems 13, (2001) 81-102

Determining the Improvement Potential of a Software Development Organization Through Fault Analysis: A Method and a Case Study

Lars-Ola Damm[1,2], Lars Lundberg[2], and Claes Wohlin[2]

[1] Ericsson AB, Ölandsgatan 1,
Box 518, SE-371 23, Karlskrona
Lars-Ola.Damm@ericsson.com
[2] School of Engineering, Blekinge Institute of Technology,
Box 520, SE-372 25 Ronneby
{Lars-Ola.Damm,Lars.Lundberg,Claes.Wohlin}@bth.se

Abstract. Successful software process improvement depends on the ability to analyze past projects and determine which parts of the process that could become more efficient. One typical data source is the faults that are reported during product development. From an industrial need, this paper provides a solution based on a measure called faults-slip-through, i.e. the measure tells which faults that should have been found in earlier phases. From the measure, the improvement potential of different parts of the development process is estimated by calculating the cost of the faults that slipped through the phase where they *should* have been found. The usefulness of the method was demonstrated by applying it on two completed development projects at Ericsson AB. The results show that the implementation phase had the largest improvement potential since it caused the largest faults-slip-through cost to later phases, i.e. 81 and 84 percent of the total improvement potential in the two studied projects.

1 Introduction

For many modern software development organizations it is of crucial importance to reduce development cost and time-to-market while still maintaining a high level of product quality. Such organizations seek specialized processes that could give them rapid improvements. However, they often overlook existing routinely collected data that can be used for process analysis [4]. One such data source is fault reports since avoidable rework accounts for a significant percentage of the total development time, i.e. between 20-80 percent depending on the maturity of the development process [10]. In fact, related research states that fault analysis is the most promising approach to software process improvement [5].

A software development department at Ericsson AB develops component-based software for the mobile network. In order to stay competitive, they run a continuous process improvement program where they regularly need to decide where to focus the current improvement efforts. However, as considered a common reality in industry, the department is already aware of potential improvement areas; the challenge is to prioritize the areas to know where to focus the improvement work [12]. Without such a decision support, it is common that improvements are not implemented because

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 138–149, 2004.

organizations find them difficult to prioritize [12]. Further, if a suggested improvement can be supported with data, it becomes easier to convince people to make changes more quickly [5]. As stated above, fault statistics is one useful information source in software process improvement; therefore, the general research question of this paper is:

> How can fault statistics be used for determining the improvement potential of different phases/activities in the software development process in an organization?

A commonly used approach for fault analysis is classification of faults from their causes, e.g. root cause analysis [9]. Although root cause analysis can provide valuable information about what types of faults the process is not good at preventing/removing, the technique is cost intensive and therefore not easy to apply on larger populations of faults. Further, the classification procedure should not require advanced skills that normally only professional analysts have; ordinary developers must be able to perform the classification [1]. Finally, and most importantly, root cause analysis does not quantify what the improvement potential of different phases is.

In order to satisfy the above given requirements, this paper introduces a method in which a central part is a 'faults-slip-through' measure. That is, the faults are classified according to whether they slipped through the phase where they should have been found. This approach to fault classification has been used before [1,6]; the main difference with the approach here is that it also calculates the improvement potential by relating the result of the fault classification with the average cost of finding and repairing the faults. From the classified faults, the method measures the improvement potential by multiplying the faults-slip-through distribution with the average benefit of finding a fault earlier. In order to verify the applicability of the method, the paper also provides an empirical case study where the method is applied on the faults reported in two finished development projects at Ericsson AB.

The paper is outlined as follows. Section 2 describes the proposed method for how to determine the improvement potential of an organization. Section 3 demonstrates the applicability of the method through an empirical case study. Section 4 discusses the validity and implications of the results and Section 5 concludes the work.

2 Method

2.1 Estimation of Improvement Potential

The purpose of this paper is to demonstrate how to determine the improvement potential of a development process from historical fault data. This section describes the selected method for how to achieve this through the following three steps:

1. Determine which faults that could have been avoided or at least found earlier
2. Determine the average cost of finding faults in different phases.
3. Determine the improvement potential from the results in (1) and (2).

In this context, a fault is defined as an anomaly that causes a failure [7].

When using fault data as basis for determining the improvement potential of an organization's development process, the essential analysis to perform is whether the faults could have been avoided or at least have been found earlier. As previously

mentioned, the introduced measure for determining this is called 'faults-slip-through', i.e. whether a fault slipped through the phase where it should have been found. The definition of it is similar to measures used in related studies, e.g. phase containment metrics where faults should be found in the same phase as they were introduced [6], and goodness measures where faults should be found in the earliest possible phase [1]. In practice, the only difference between the faults-slip-through definition and the other definitions is when a fault is introduced in a certain phase but it is not efficient to find in the same phase, e.g. a certain test technique might be required to simulate the behaviour of the function. Table 1 provides a fictitious example of faults-slip-through between arbitrarily chosen development phases. The columns represent in which phase the faults were found (PF) and the rows represent where the faults should have been found (Phase Belonging, PB). For example, 25 of the faults that were found in Function Test should have been found during implementation (e.g. through inspections or unit tests). Further, the rightmost column summarizes the amount of faults that belonged to each phase whereas the bottom row summarizes the amount of faults that were found in each phase. For example, 49 faults belonged to the implementation phase whereas most of the faults were found in Function Test (50).

Table 1. Fictious example of faults-slip-through data (nr. faults found, belonging /phase)

PF: / PB:	Design	Impl.	Function Test	System Test	Operation	**Total belonging/phase**
Design	1	1	10	5	1	**18**
Impl.		4	25	18	2	**49**
Function Test			15	5	4	**24**
System Test				13	2	**15**
Operation					0	**0**
Tot. found/phase	**1**	**5**	**50**	**41**	**9**	**106**

When having all the faults categorized according to the faults-slip-through measure, the next step is to estimate the cost of finding faults in different phases. Several studies have shown that the cost of finding and fixing faults increases more and more the longer they remain in a product [3, 10]. However, the cost-increase varies significantly depending on the maturity of the development process and on whether the faults are severe or not [10]. Therefore, the average fault cost in different phases needs to be determined explicitly in the environment where the improvement potential is to be determined (see a fictitious example in Table 2). This measure could either be obtained through the time reporting system or from expert judgments, e.g. a questionnaire where the developers/testers that were involved in the bug-fix give an estimate of the average cost.

Table 2. Fictious example of average fault cost/phase found

	Design	Implementation	Function test	System test	Operation
Average fault cost	1	2	10	25	50

Expert judgments are a fast and easy way to obtain the measures; however, in the long-term, fully accurate measures can only be obtained by having the cost of every

fault stored with the fault report when repairing it. That is, when the actual cost is stored with each fault report, the average cost can be measured instead of just being subjectively estimated. Further, when obtaining these measures, it is important not just to include the cost of repairing the fault but also fault reporting and re-testing after the fault is corrected.

The third step (e.g. the improvement potential) is determined by calculating the difference between the cost of faults in relation to what the fault cost would have been if none of them would have had slipped through the phase where they were supposed to be found. Fig. 1 provides the formulas for making such a calculation and as presented in the table in the figure, the improvement potential can be calculated in a two-dimensional matrix. The equation in the figure provides the actual formula for calculating the improvement potential for each cell (IP_{xx}). $PF_{x\ total}$ and $PBx_{\ total}$ are calculated by summarizing the corresponding row/column. As illustrated rightmost in the figure, the average fault cost (as discussed in the previous paragraph), need to be determined for each phase before using it in the formula (IP_{xx}). In order to demonstrate how to use and interpret the matrix, Table 3 provides an example calculation by applying the formulas in Fig. 1 on the fictitious values in Table 1 and Table 2.

	PF_1	PF_2	PB_{total}
PB_1	IP_{11}	IP_{12}	$PB_{1\ total}$
PB_2	IP_{21}	IP_{22}	$PB_{2\ total}$
PF_{total}	$PF_{1\ total}$	$PF_{2\ total}$	$(PB/PF)_{total}$

AvFC
PB_1 AvFC
PB_2 AvFC

PF = Phase found, PB =Phase belonging, AvFC=Average Fault Cost
IP = Improvement potential

IP_{xx} = (Nr faults bel. (PBx) * PB_1 AvFC) – (Nr faults bel. (PBx) * PBxAvFC)

Fig. 1. Matrix formula for calculation of improvement potential

In Table 3, the most interesting cells are those in the rightmost column that summarizes the total cost of faults in relation to fault belonging and the bottom row that summarizes the total unnecessary cost of faults in relation to phase found. For example, the largest improvement potential is in the implementation phase, i.e. the phase triggered 710 hours of unnecessary costs in later phases due to a large faults-slip-through from it. Note that taking an action that removes the faults-slip-through from the implementation phase to later phases will increase the fault cost of the implementation phase, i.e. up to 49 hours (1 hour/fault times 49 faults that according to Table 2 belonged to the implementation phase). Further, System Test is the phase that suffered from the largest excessive costs due to faults slipped through (609 hours). However, when interpreting such excessive costs, one must be aware of that some sort of investment is required in order to get rid of them, e.g. by adding code inspections. Thus, the potential gain is probably not as large as 609 hours. Therefore, the primary usage of the values is to serve as input to an expected Return On.

Investment (ROI) calculation when prioritizing possible improvement actions.

When measured in percent, the improvement potential for a certain phase equals the improvement potential in hours divided with the total improvement potential (e.g. in the example provided in Table 3, the improvement potential of System Test = 609/1255=49%). In the case study reported below, the measurements are provided in percent (due to confidentiality reasons).

Table 3. Example of calculation of improvement potential (hours)

PF: / PB	Design	Impl.	Function Test	System Test	Operation	**Total PB/phase**
Design	1*1-1*1 = 0	1*2-1*1 = 1h	10*10-10*1 = 90h	5*25-5*1 = 120	1*50-1*1 = 49	**260h**
Impl.		4*2-4*2 = 0	25*10-25*2 = 200h	18*25-18*2 = 414h	2*50-2*2 = 96h	**710h**
Function Test			15*10-15*10 = 0	5*25-5*10 = 75h	4*50-4*10 = 160h	**235h**
System Test				13*25-13*25 = 0	2*50-2*25 = 50h	**50h**
Operation					0	**0h**
Total potential/ PF		**1h**	**290h**	**609h**	**355h**	**1255h**

In related work, calculations on the improvement potential from faults have been applied by calculating the time needed in each phase when faults were found when supposed to in comparison to when they were not [11]. Although the results of using such an approach are useful for estimating the effect of implementing a certain improvement, they require measurements on the additional effort required for removing the faults earlier. Such measurements require decisions on what improvements to make and estimates of what they cost and therefore they cannot be used as input when deciding in which phases to focus the improvements and what the real potential is.

3 Results from Applying the Method

The applicability of the described method was evaluated by using it on the faults reported in two projects at a department at Ericsson AB. The projects developed new functionality to be included in new releases of the two different products. Hence, previous versions of the products were already in full operation at customer sites. Further, the projects used the same processes and tools and the products developed in the projects were developed on the same platform (i.e. the platform provides a component-based architecture and a number of platform components that are used by both products). The products were developed mainly in C++ except for a Java-based graphical user interface that constitutes minor parts of each product. Apart from the platform components, each product consists of about 10-30 components and each component consists of about 5-30 classes. The reason for studying more than one project was to be able to strengthen the validity of the results, i.e. two projects that were developed in the same environment and according to the same development process should provide similar results (except for eventual known events in the projects that affected the results). Further, two projects were chosen since the selected projects were the only recently finished projects and because earlier finished projects were not developed according to the same process. Thereby, it was the two selected projects that could be considered as representative for the organization.

The reported faults originated from the test phases performed by the test unit at the department, i.e. faults found earlier were not reported in a written form that could be post-analyzed. Further, during the analysis, some faults were excluded either because

they were rejected or because they did not affect the operability of the products, e.g. opinion about function, not reproducible faults, and documentation faults.

3.1 Faults-Slip-Through

Fig. 2 and Fig. 3 present the average percent faults-slip-through in relation to percent faults found and development phase from two finished projects at the department. The faults-slip-through measure was not introduced until after the project completions, and hence all the fault reports in the projects studied needed to be classified according to the method described in Section 2 in retrospect. The time required for performing the classification was on average two minutes/fault. Actually, several faults could be classified a lot faster but some of them took a significantly longer time since these fault reports lacked information about the causes of the faults. In those cases, the person that repaired the fault needed to be consulted about the cause. Additionally, in order to obtain a consensus on what faults should be considered as faults-slip-through and not, a workshop with key representatives from different areas at the department was held. The output from the workshop was a checklist specifying which faults that should be considered to belong to which phase. When assigning faults to different phases, the possible phases to select among were the following:

System Design (SD): Faults in the design of the external interfaces of the product.

Implementation (Imp): Faults found when implementing the components, e.g. low-level design and coding faults as well as faults found during inspections and unit tests.

Integration Test (IT): Faults found during primary component integration tests, e.g. installation faults and basic component interaction faults.

Function Test (FT): Faults found when testing the features of the system.

System Test (ST): Includes integration with external systems and testing of non-functional requirements.

Field Test + 6 Months (FiT+6): During this period, the product is tested in a real environment (e.g. installed into a mobile network), either at an internal test site or together with a customer. During the first six months, most issues should be resolved and the product then becomes accepted for full operation.

Field Test 7-12 Months (FiT_7-12): Same as FiT+6; however, after 6 months of field tests, live usage of the product has normally begun.

As can be seen in the figures, several faults belonged to the implementation phase (59, 66%). Further, in project A (Fig. 3), many faults were found in FiT+6 (29%). The primary reason for this was that the field tests started before ST was completed, i.e. the phases were overlapping which resulted in that ST continued to find faults during FiT+6. These ST faults could for practical reasons only be classified as FiT+6 faults.

3.2 Average Fault Cost

When estimating the average fault cost for different phases at the department, expert judgments were used since neither was the fault cost reported directly into the fault reporting system nor was the time reporting system feasible to use for the task. In

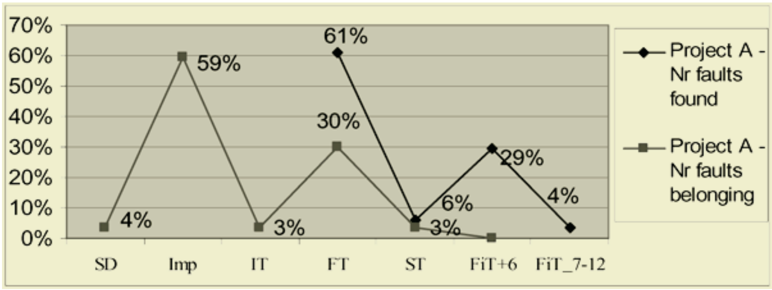

Fig. 2. Percent faults-slip-through in relation to percent faults found and development phase (Project A)

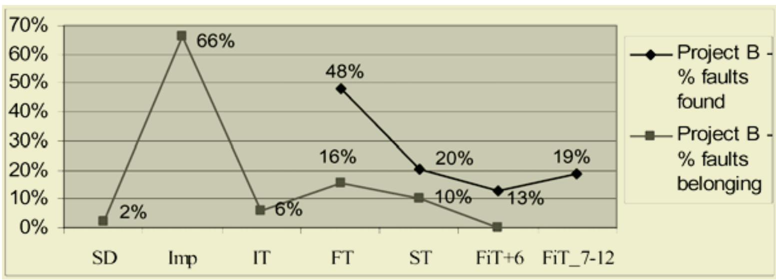

Fig. 3. Percent faults-slip-through in relation to percent faults found and development phase (Project B)

practice, this means that the persons that were knowledgeable in each area estimated the average fault cost. Table 4 presents the result of the estimations. For example, a fault costs 16 times more in System Test (ST) than in System Design (SD).

Table 4. Average fault-cost/phase at the department (in relative terms)

Phase found	SD	Imp	IT	FT	ST	FiT+6	FiT_7-12
Average cost/fault	1	1.2	3	9.8	16	20	20

The performed cost estimations only include the time required for reporting, fixing and re-testing each fault, which means that there might be additional costs such as the cost of performing extra bug-fix deliveries to the test department. Such a cost is hard to account for since the amount of deliveries required is not directly proportional to the amount of faults, i.e. it depends on the fault distributions over time and the nature of the faults. The reason why FiT_7-12 was estimated to have the same cost as FiT+6 was because the system was still expected to be in field tests although live usage in reality actually might already have started. Further, during the first 12 months after the field tests have started, few systems have been installed although the system becomes available for live usage already during this period. That is, the fault cost rises when more installed systems need to be patched, but, in reality, this does not take any effect until after FiT_7-12.

3.3 Improvement Potential

Table 5 and Table 6 present the improvement potential of the two studied projects from the fault statistics provided in Sections 3.1 and 3.2 , calculated according to the method provided in Section 2. As can be seen in both tables, faults-slip-through from Implementation comprised a significant proportion of the improvement potential (81%, 84%); therefore, this is foremost where the department should focus their improvement efforts. Further, in project B, all the test phases had a significant improvement potential, e.g. FT could be performed at a 32% lower cost by avoiding the faults-slip-through to it. On the contrary, project A had more diverse fault distributions regarding phase found. The reason for this is mainly due to overlapping test phases (further discussed in Section 4). Finally, it should also be noted that the total improvement potential in relation to fault origin phase (rightmost columns) are similar for both projects, which strengthens the assumption in that the improvement potential is foremost process related, i.e. the faults-slip-through did not occur due to certain product problems or accidental events in the projects.

Table 5. Improvement potential (Project A)

Phase found Phase belonging	FT	ST	FiT+6	FiT_7-12	Total potential /origin phase
SD	3.3%	0.0%	0.6%	0.0%	**3.9%**
Imp	33%	5.6%	37%	5.4%	**81%**
IT	2.2%	0.0%	0.5%	0.5%	**3.2%**
FT	0.0%	0.7%	10.1%	0.6%	**11%**
ST	0.0%	0.0%	0.8%	0.1%	**0.9%**
Total potential/test phase	**39%**	**6.3%**	**48%**	**6.6%**	**100%**

Table 6. Improvement potential (Project B)

Phase found Phase belonging	FT	ST	FiT+6	FiT_7-12	Total potential /origin phase
SD	0.6%	1.9%	0.0%	0.0%	**2.5%**
Imp	30%	14%	12%	28%	**84%**
IT	1.7%	2.5%	2.2%	0.0%	**6.4%**
FT	0.0%	1.6%	1.3%	2.0%	**4.9%**
ST	0.0%	0.0%	1.5%	0.8%	**2.3%**
Total potential/test phase	**32%**	**20%**	**17%**	**30%**	**100%**

4 Discussion

4.1 Validity Threats to the Results

When conducting an empirical industry study, the environment cannot be controlled to the same extent as in isolated research experiments. In order to be able to make a correct interpretation of the results presented in Section 3, one should be aware of

threats to the validity of them. As presented below, the main validity threats to this case study concern conclusion, internal, and external validity [8]. Construct validity is not relevant in this context since the case study was conducted in an industrial setting.

Conclusion validity concerns whether it is possible to draw correct conclusions from the results, e.g. reliability of the results [8]. The threats to conclusion validity are as follows. First, when determining which phase each fault belonged to, several faults were assigned to the implementation phase. However, some of these faults might as well belong to System Design due to a lack of information on the causes of the faults. That is, from a described fault cause, it was possible to determine that the fault should have been found before testing started but determining whether a fault belonged to system design or to implementation was sometimes hard since the fault description did not state if the fault occurred due to a fault in a specification or in the implementation. Nevertheless, the practical effect of this uncertainty was small since the cost of finding faults in these two phases was estimated as almost the same (see Table 3). Second, in order to be able to draw conclusions from the results, the department must have a common view on which phase each fault should belong to. That is, managers and developers should together agree on which fault types that should be considered as faults-slip-through and not. At the studied department, this was managed by having workshops where checklists for how to estimate fault-slip-through were developed. However, continuous improvements and training are required in the first projects in order to ensure that everyone have the same view on how to make the estimations. Further, regarding the average fault cost for different phases, the result was obtained through expert judgments and therefore, the estimations might not exactly reflect the reality. However, this was minimized by asking as many ‘experts’ as possible, i.e. although there might be deviations, the results were good enough to measure the improvement potential from and hence use as basis for decisions. However, in the future, direct fault cost measures should be included in the fault reports so that this uncertainty is removed. Finally, since the improvement potential is calculated from the faults-slip-through measure and the average fault cost measure, the accuracy of the improvement potential is only dependent on the accuracy of the other measures.

Internal validity concerns how well the study design allows the researchers to draw conclusions from causes and effects, i.e. causality. For example, there might be factors that affect the dependent variables (e.g. fault distributions) without the researchers knowing about it [8]. In the case study presented in this paper all faults were post-classified by one researcher, which thereby minimized the risk for biased or inconsistent classifications. Another threat to internal validity is whether certain events that occurred during the studied projects affected the fault distribution, i.e. events that the researchers were not aware of. This was managed through workshops with project participants where possible threats to the validity of the results were put forward. Additionally, since two projects were measured, the likelihood of special events that affected the results without being noticed decreased.

External validity concerns whether the results are generalizable or not [8]. In this case study, the results are in overall not generalizable since they are only valid for the studied department having certain products, processes, and tools. However, since two projects were studied and gave similar fault distributions, the results are generalizable within the context of the department. Further, the results on average fault costs in different phases (see Table 4) acknowledge previous claims in that faults are significantly more expensive to find in later phases [3, 10]. Nevertheless, in this paper, the

main concern regarding external validity is whether the method used for obtaining the results is generalizable or not. Since the method contains no context dependant information, there are no indications in that there should be any problems in applying the method in other contexts. Thus, the method can be replicated in other environments.

4.2 Implications of the Results

The primary implication of the results is that they provide important input when determining the Return On Investment (ROI) of process improvements. That is, since software process improvement is about reducing costs, the expected ROI needs to be known; otherwise, managers might not want to take the risk to allocate resources for handling upfront costs that normally follow with improvements. Additionally, the results can be used for making developers understand why they need to change their ways of working, i.e. a quantified improvement potential motivates the developers to cooperate [11].

Improvement actions regarding the issues resulting in the largest costs were implemented in subsequent projects, e.g. more quality assurance in earlier phases. Besides shortening the verification lead-time, the expected result of decreased faults-slip-through percentages was to improve the delivery precision since the software process becomes more reliable when many defects are removed in earlier phases [11]. The projects using the method will be studied as they progress.

An unanticipated implication of introducing the faults-slip-through measure was that it became a facilitator for discussing and agreeing on what to test when, e.g. improvement of test strategies. This implies that the measure could serve as a key driver for test process improvement.

5 Conclusions and Further Work

The main objective of this paper was to answer the following research question:

> How can fault statistics be used for determining the improvement potential of different phases/activities in the software development process in an organization?

The answer to the research question constitutes a new method for determining the improvement potential of a software development organization. The method comprises the following three steps:

(1) Determine which faults that could have been avoided or at least found earlier, i.e. faults-slip-through.

(2) Determine the average cost of faults found in different phases.

(3) Determine the improvement potential from the measures in (1) and (2), i.e. measure the cost of not finding the faults as early as possible.

The practical applicability of the method was determined by applying it on two industrial software development projects. In the studied projects, potential improvements were foremost identified in the implementation phase, e.g. the implementation phase inserted, or did not capture faults present at least, too many faults that slipped through to later phases. For example, in the two studied projects, the Function Test phase

could be improved by up to 32 and 38 percent respectively by decreasing the amount of faults that slipped through to it. Further, the implementation phase caused the largest faults-slip-through to later phases and thereby had the largest improvement potential, i.e. 81 and 84 percent in the two studied projects.

The measures obtained in this report provide a solid basis for where to focus improvement efforts. However, in further work, the method could be complemented with investigations on causes of why faults slipped though the phase where they should have been found. For example, the distribution of faults-slip-through could be related to different software modules in order to be able to focus more efforts on modules that have a high faults-slip-through. Further, when using the method described in this paper, it would be very interesting to be able to quantify what effect the provided measures have on the development lead-time and delivery precision, e.g. through fault prediction models. To clarify, it is quite obvious that an organization that has a high improvement potential plausibly also has a longer development lead-time than necessary. Further, when having a longer verification lead-time, the organization cannot be as certain about when the product will reach an adequate quality level and thereby is ready to be delivered. The challenge is to be able to quantify this relation.

Acknowledgements

This work was funded jointly by Ericsson AB and The Knowledge Foundation in Sweden under a research grant for the project "Blekinge – Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

References

1. Berling, T., Thelin, T., An Industrial Case Study of the Verification and Validation Activities, Proceedings of the Ninth International Software Metrics Symposium, IEEE, (2003) 226-238
2. Bhandari, I., Halliday, M., Tarver, E., Brown, D., Chaar, J., Chillarege, R., A Case Study of Software Process Improvement During Development, IEEE Transactions on Software Engineering, Vol. 19, Issue 12, Proquest (1993) 1157-1171
3. Boehm, B., Software Engineering Economics, Prentice-Hall (1981)
4. Cook, E., Votta, L., Wolf, L., Cost-Effective Analysis of In-Place Software Processes, IEEE Transactions on Software Engineering, Vol. 24, Issue 8, Proquest (1998) 650-662
5. Grady, R., Practical Software Metrics for Project Management and Process Improvement, Prentice Hall (1992)
6. Hevner, A. R., Phase Containment for Software Quality Improvement, Information and Software Technology 39 (1997) 867-877
7. IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE/ANSI Standard 982.2-1988
8. Wohlin, C., Höst, M., Henningsson, K., Empirical Research Methods in Software Engineering, In Empirical Methods and Studies in Software Engineering: Experiences from ESERNET, pp. 7-23, editors Reidar Conradi and Alf Inge Wang, Lecture Notes in Computer Science, Spinger-Verlag, Germany, LNCS 2765
9. Leszak, M., Perry, D., Stoll, D., A Case Study in Root Cause Defect Analysis, Proceedings of the 22nd Int. Conference on Software Engineering, ACM Press, (2000) 428-437

10. Shull, F., Basili V., Boehm B., Brown W., Costa, P., Lindwall, M., Port, D., Rus, I., Tesoriero, R., Zelkowitz, M., What We Have Learned About Fighting Defects, Proceedings of the Eight IEEE Symposium on Software Metrics, IEEE (2002) 249-258
11. Tanaka T., Sakamoto K., Kusumoto, S., Matsumoto K., Kikuno T., Improvement of Software Process by Process Description and Benefit Estimation, Proceedings of the 17th International Conference on Software Engineering, ACM (1995) 123-132
12. Wohlwend H., Rosenbaum S., Software Improvements in an International Company, Proceedings of the 15th International Conference on Software Engineering, IEEE Comput. Soc. Press. (1993) 212-220

Root Cause Analysis and Gap Analysis – A Tale of Two Methods

Tor Stålhane

NTNU
Norwegian University of Science and Technology
stalhane@idi.ntnu.no

Abstract. We performed a gap analysis in an SME in order to identify SPI opportunities. One month later, we also performed a root cause analysis, based on error report data, in the same company and with the same developers. Although the two approaches are quite different, the end results turned out to be rather similar. We should be careful when drawing general conclusions from one single case study but the study indicates that for SPI, the participants' problems, preferences and experiences are more important than the data collected in the start-up phase.

1 Introduction

The opportunity to write this paper stems from a quite unplanned activity in the Norwegian SPI program SPIQ [1] some years ago. Two teams of researchers had, quite unintentionally and only a month apart, done some SPI work together with the developers in a Norwegian SME. One of the research teams first used a Pareto analysis on the error report statistics and then a root cause analysis (RCA) to identify improvement opportunities, while the other team used a gap analysis followed by a brainstorming session. Since all material from both groups is available, it is interesting to look at two research questions:

1. To what extent did the two approaches give the same results?
2. If we find large differences, what are their main causes?

We expected to find large differences – after all, the two SPI activities employed dramatically different approaches and had different foci – future goals versus current problems.

2 The Methods Used

2.1 Pareto Analysis

When using the RCA, we need to identify a starting point. In order to attack the most frequent error causes, we started by using a Pareto analysis. Pareto analysis [2] is

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 150–160, 2004.

named after the Italian economist Vilfredo Pareto, who discovered that 30% of the families in Italy owned 70% of all wealth in the country. The Pareto principle is also called the principle of the important few and the insignificant many. This principle has turned out to have wide range of applications – 30% of the code in the system takes care of 70% of the input, 30% of the errors are responsible for 70% of the rework costs and so on. The Pareto principle is not a law of nature; it is just a principle that helps us to organize our observations.

If we want to know if the Pareto rule holds for our observations, we can sort the observations according to their values – largest value first – and plot the data. If the plot is concave, Pareto's rule might hold. Compute the sum of all the factor values – S – and the value of the largest third of the factor values – T. If we find that T is close to 0.7*S, we probably have a Pareto distribution.

2.2 Root Cause Analysis – RCA

The root cause analysis [2] – RCA – is usually one of the last steps in the process of identifying improvement opportunities. This does not imply that we always need the RCA. In many cases, improvement opportunities can be identified during a GQM feedback meeting [3] or in the post-it notes session of a post mortem analysis (PMA) [4]. The setting is usually that we have identified a problem and want to find its root causes.

The RCA diagram is also called the fishbone diagram for obvious reasons. When constructing the Ishikawa diagram, we start with the problem we want to analyze at the right hand side of the diagram and a horizontal line pointing to the problem. We then ask, "What can be the causes of this problem?" The answers to this question, the first order causes, are included in the diagram with lines – fish bones – pointing towards the main line – the spine of the fish. We can then repeat the question of causes for each of the first order causes and so on for as long as we feel necessary.

2.3 Gap Analysis

A gap analysis [5] builds on two pieces of information – where are we and where do we want to be? There are several ways to do such an analysis, but the simplest way is to use the following form:

Present strength – P					**Property**	**Future importance – F**				
1	**2**	**3**	**4**	**5**		**1**	**2**	**3**	**4**	**5**

Each participant gets his own copy of the form to fill in. The results are summed up in a bar chart – one set of bars for present strength and one set of bars for the future importance. The bar charts are used as input to a feedback meeting. It is the participants who should interpret the results. Even though it is a slight violation of the

science of statistics to do so, we compute the average values for the strength and future importance by treating the score values as values on an ordinal scale. We find the priorities of the properties by sorting the table according to the differences between the two averages. The final priorities must, however, be decided by the participants. With the notation used in the table above, we can write the following expression:

$$\sum_i (F_i - P_i)/(number\_of\_items) \tag{1}$$

Gap analysis is simple, un-bureaucratic and efficient. It is useful for getting a first set of priorities and setting the future course for the company's SPI activities.

3 The Main Results – SPI Opportunities

3.1 Pareto and RCA

In this approach, the log of errors reported from the customer was our starting point. The reports from the error-logging tool combined with a Pareto analysis gave us the graph shown in Fig. 1. We see that the number of person days used for error correction is Pareto distributed. The graph is concave. We have six categories and of these the first third – two categories – are responsible for 155 person days or 79% of all person days used for error correction. In order to limit the amount of SPI opportunities, we decided to only look at the most important category – logical problems. Pareto's rule holds also for the distribution of causes for logical problems. Again the graph is concave and a third of the categories – two categories – hold 81% of all person days used for error correction.

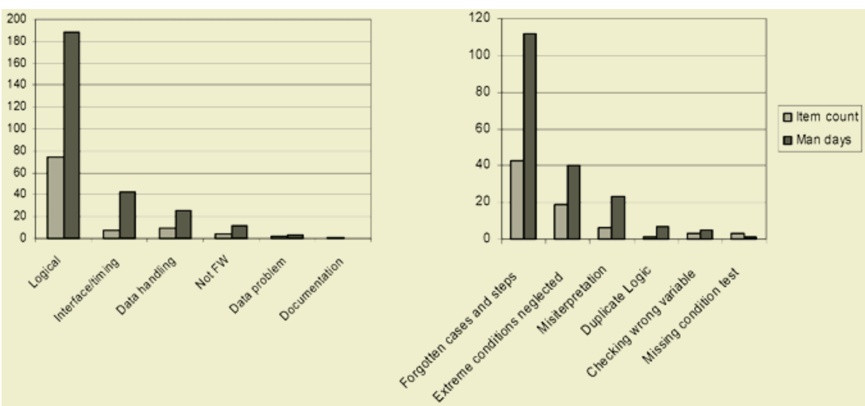

Fig. 1. Distribution of problem sources and correction resources

Fig. 1 shows on the left side that logical problems are the most costly error cause for errors. On the right side in Fig. 1 we see that "forgotten cases and steps" and "extreme conditions neglected" are the most frequent cause for this error category. Thus, in the RCA we focused on these two problem areas.

All the developers participated in a RCA to identify the main causes for these two error categories. The Ishikawa diagram for the first one of the two RCA analyses is shown in Fig. 2 below. Each participant scored the main causes in both Ishikawa diagrams according to their importance. The most important error causes from the two Ishikawa diagrams were selected based on the sums of scores for each cause. This gave the following results:

- Forgotten cases and steps
  - Loss of oversight – how the customers use the product, lack of competence transfer when people quit and too many states in the software
  - Time pressure – quick design solutions and management doesn't want to set priorities.
  - Old code is part of many new products – stable products versus new development and too many changes.
- Extreme conditions neglected
  - Lacking important competence – product design should reflect customers' needs and too complex code
  - Error handling.
  - Lack of design control – incomplete design before coding and the design solutions are not flexible enough.

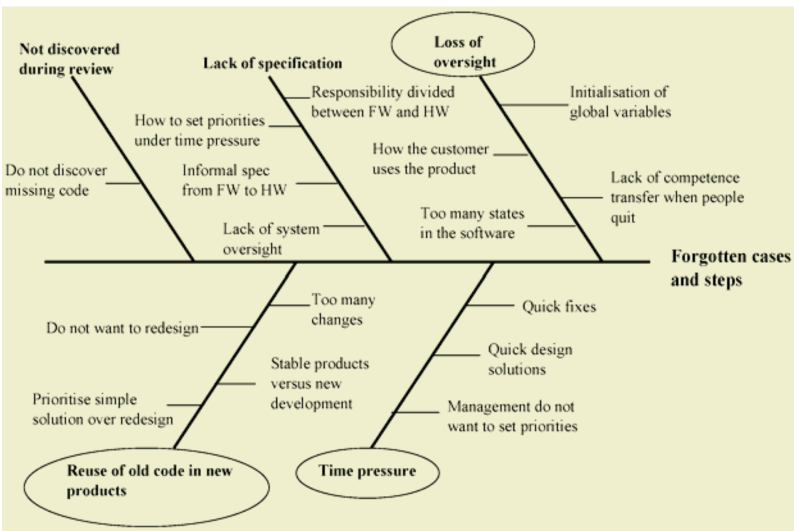

Fig. 2. Ishikawa diagram for "Forgotten cases and steps"

The problem causes from the Ishikawa diagram were used as a starting point and a set of foci for a brainstorming session. As a result of this, we created the table of SPI activities shown in Table 1 together with the developers.

All activities that received the votes of more than half of the participants – more than three votes – were selected for implementation. If several activities received the same number of votes, they all got the average rank of the activities with the same

Table 1. Results from the RCA analysis – first phase

Activity	Means	Votes	Priority
Focus on the right job – better, more consistent prioritizing		6	1
Working module groups with assign responsibilities	Study groups	5	2.5
Simplify and reduce the retry-algorithm for writing to tape		2	7.5
Better documentation of error situations	Define the error situations during specification	1	10
Use experiences from several disciplines			12.5
Increase observability of error situations	Better tracing	3	5
More people to participate in the design phase		1	10
Better coordination between development and maintenance		1	10
Information development	Cross-project reviews	2	7.5
Better motivation	Need to understand what motivates the developers	3	5
More communication between marketing and development		3	5
Reduce the levels of ambition for functionality – it is risky to change old code	Priorities functionality and perform a risk analysis	5	2.5
Improved product testing		1	12.5

vote. When we compare the output from the RCA and the gap analysis, we will only consider the six activities with the highest number of votes.

3.2 Gap Analysis

The questions used in the gap analysis are adapted from an ESI survey from 1995 [5]. This analysis focuses on three main areas – competition, life cycle processes and learning and process improvement – see the main categories in tables 3 and 4. In addition to the questions from this survey, the participants also included some new, company specific questions. All personnel in the group – developers, project and group managers and marketing personnel – participated in the gap analysis. The areas in the gap analysis were ranked according to their average shifts – see equation (1). The bigger the average shift, the more important is the SPI activities that can be derived from it. Most of the time, all three groups agreed on the size of the shifts.

The average shift value was not, however, the only input to the prioritizing process. In addition, the participants took the company's internal priorities into consideration. For this reason, not all areas with large shifts got a top priority and some areas

with a smaller shift – 1.4 and 1.5 – got a top priority ranking. Based on the results from the gap analysis, the company decided on the SPI activities shown in Table 2.

Table 2. Results from the gap analysis – first phase

Problem area	Means	Priority
Training of new employees	The training of new employees must start while those that are about to be replaced are still working	1
Time pressure Fail to deliver on time	Marketing must not promise a delivery data without an OK from the development group leaders	2
Cooperation, and discussions when writing specifications for new products	Review of the Overland case. What went wrong – and why?	3
How do the customers use the system?	Internal seminars to discuss how the systems are used by the customers Test the equipment on customer hardware	4
Increase the flexibility of configuration management	New procedures for configuration management New procedures for handling versioning	5
Redesign of error prone modules	Identify error prone modules Decide whether we will do a complete redesign or redesign only for new products.	6

The results in the table come from a combination of the output from the gap analysis and the brainstorming session. The participants had the results from the gap analysis with them during this session.

4 Comparing the Two Methods

The two methods have more in common than we thought at first glance. The first thing to note is that both methods have a strong subjective component. The last step is the same for both methods – we use personal experience plus information from earlier parts of the process to identify improvement opportunities.

We will look at the differences between the methods at two stages in the process – after the data collection and after we have identified the SPI opportunities. The results from the data collection in the gap-analysis and the RCA are shown in Table 3. In order to keep it simple, we used the categories from the gap-analysis as table categories. The gap-analysis data can then be inserted directly, while we need some interpretation to decide the right row for the RCA data. In order to limit the size of the table and to facilitate comparison with the RCA data, we used only the data supplied by the developers and excluded the data from the management and marketing personnel.

Table 3. Results from RCA and gap analysis – first phase

Categories	Gap analysis	RCA
Competition		
Market	–	–
Flexibility	–	–
Quality	Prioritize long-term quality before quick problem fixing To do the job correctly the first time irrespectively of time pressure	Quick design solution Incomplete design before coding
Deliveries	–	–
Life cycle processes		
Delivery processes	Marketing should know the customers' use of the product Developers should know the customers use of the product	How the customers use the product Product design – should reflect customers' needs
Development processes	–	Too complex code Too many states in the software Too little focus on error handling in the product
Support processes	–	Too many changes
Management processes	The developers get the resources they need to meet their goals	Management do not want to set priorities
Organizational processes	Courses necessary to allow each employee to do his job Mechanisms that support cooperation between departments Mechanisms that support cooperation between disciplines	Lack of competence transfer when people quit
Learning and process improvement		
Learning in the group	–	–
Learning from other groups	–	–
Software process improvement – SPI	Testing of new technical knowledge in new projects	Stable products versus new development

Table 3 shows two things – firstly, the gap analysis and the RCA mostly agree on what are the important issues. If we for instance look at the category "quality", we see that both methods identify time pressure as an important quality problem. The same agreement is found for "delivery processes", "management processes" and "software process improvement". The exceptions are the categories "support process" and "development process", which are not included in the results from the gap-analysis. The reason for this is that these items have a high score for present strength (mostly above 3.0) and it is thus impossible to get a gap larger than 3.0 for any of these items. The differences are as should be expected when using two rather different methods and

foci. The agreement on "organizational processes" is only partial as the RCA did not identify cooperation as an important issue.

The next phase, where the developers used brainstorming to identify SPI opportunities, brought some rather interesting changes – see Table 4.

Table 4. Results from RCA and gap analysis – second phase

Categories	Gap analysis	RCA
Competition		
Market	–	–
Flexibility	–	–
Quality	Time pressure, fail to deliver on time	–
Deliveries	–	–
Life cycle processes		
Delivery processes	Cooperation and discussion when writing specifications for new products How do the customers use the system	More communication between marketing and development
Development processes	Redesign of error prone modules	Increase observability of error situations Reduce the level of ambitions for functionality
Support processes	Increase the flexibility of the configuration management system	–
Management processes	–	Better, more consistent prioritizing Better motivation
Organizational processes	Training of new employees	–
Learning and process improvement		
Learning in the group	–	Working module groups with assigned responsibilities
Learning from other groups	–	–
Software process improvement – SPI	–	–

RCA: The items related to the "quality" category – time pressure – are gone, together with the concern for too large change traffic in the "support processes", and the problems pertaining to the SPI category – stable products versus the need to try out new methods. The focus for learning has moved from competence transfer – part of the "organizational processes" to the need for working groups with assigned responsibilities – "learning in the group" – a concept not mentioned earlier in the RCA process. The working groups as a means of learning were, however, used earlier in the company but were removed from the organization due to lack of developer interest. In the

category “delivery processes” RCA proposed communication with marketing as a solution to the previously mentioned problem – to reflect customer needs. The same movement towards a practical solution can be seen in the category “development processes”. Here the RCA propose solving the general problem of complex code by increasing the observability error situations.

Gap Analysis: The concern for developer resources in the “management processes” is gone, together with the SPI concern for formal mechanisms that can be used to test new technologies in the development of new products. The latter is consistent with the change in the RCA results. The most interesting observation for the gap analysis is, however, the inclusion of two new items – configuration management and error prone modules. Neither of these two items is mentioned in the form that the participants in the gap analysis used. They are, however, concrete solutions to real problems. In the “delivery processes” category, the gap analysis shifts from general mechanisms used to support cooperation to the specific need for cooperation when writing specifications for new products – again a low level concrete solution to a high level abstract goal.

All changes in the RCA and the gap analysis can be explained by the need to move from abstract goals and concrete problems to concrete solutions. Problems where the developers saw no clear solution were removed. What these few data seem to say is that the more the data are transformed in people-intensive, creative processes, the more the real problems dominate, irrespective of the method used to collect the data in the first phase of the process. The diagram shown in figure 3 shows the mechanisms that are at work in both processes.

The diagram illustrates how the choice of method influences the subset of problems considered after the first phase. Thus, the results here will for instance depend on whether we use an RCA or a gap analysis. In the next phase, however, personal priorities are brought in and will be mixed with – and later dominate – the previously selected subset of problems. In the end, personal priorities of the participants will dominate the output and thus the identified improvement opportunities.

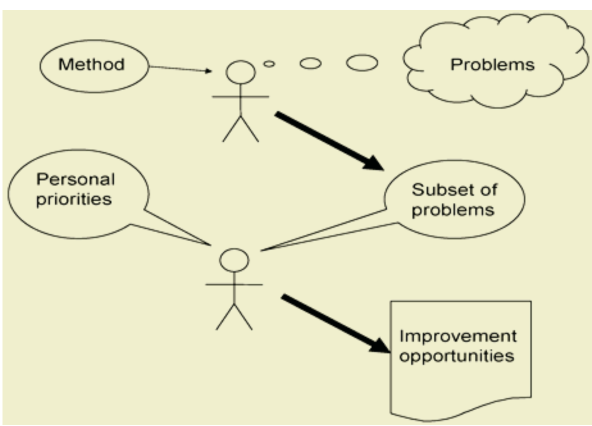

Fig. 3. The influence of methods used and perceived problem in an SPI process

We have at least two scenarios that can give the observed effect, characterized by a strong-headed consultant or by strong-headed developers respectively:

1. The consultant steers the process in the preferred direction – he has his standard approach and the developers are moved towards this – nolens volens. A discussion with the consultant that did the work in our case suggests that we can ignore this possibility, since they had no preferred solution.
2. The developers have decided what their most important problems are and try to solve these, irrespective of method and available data. Problems that cannot be solved in the near future are removed. A good consultant will in this case function as a catalyst for the solution.

The main question seems to be – who sets the course and who is at the helm? In our case, the researchers that collected the data claim that they had no preferred solution or direction of results. Thus, we are left with scenario 2 – the developers were in charge of the process and the results reflected in both cases their problems and needed solutions. For a further discussion of this and related problems in an industrial setting, see [6].

5 Conclusions

What is reported here is just one small case study and we should take care when extrapolating the results. With this caveat in mind, it seems that:

- The SPI method used will mainly decide the data that will be collected.
- The differences between any two methods will decrease with the number of person-intensive steps, for instance brainstorming activities.
- The ideas, experiences and opinions of the participants have a larger impact on the identified opportunities than the method used. We should thus – ceteris paribus – select the SPI method that creates the largest commitment in the company.

In some sense the results show the triumph of real problems over collected data. Our results give three important insights:

- The participants will use the data as input and as a starting point, but the SPI recommendations that come out in the end will in either case be a combination of the data and the company's main problems and priorities.
- It is important to create a forum where SPI researchers and developers can exchange experience and information. It is important that the developers are in charge of this forum.
- If we are trying to identify improvement opportunities in a certain area, we need to work hard to keep the focus throughout the process – for instance by using a focused PMA.

Finally, whatever process we choose for our SPI work, the process creates a forum where developers and consultants meet to solve the problem at hand. My own industrial experience has shown that an important component in this approach is that the process works to legitimize a solution that the developers have already decided that

they want. The process is needed because management is more willing to listen to the consultants than to their own developers.

Acknowledgements

I want to thank Professor M. Levin (NTNU) for valuable discussion on the people-angle of the problems and results discussed in this paper and T. Dybå and N.B. Moe (SINTEF) for letting me use their data from the gap analysis.

References

1. The SPIQ Handbook. Tapir, NTNU, Trondheim, 2001
2. D. Straker: "A Toolbook for Quality Improvement and Problem Solving", Prentice Hall, 1995. ISBN 0-13-746892-X
3. R. van Solingen and E. Berghout: "The Goal/Question/Metric Method", McGraw-Hill Companies, London, 1999, ISBN 007 709553 7
4. T. Stålhane, A. Birk and T. Dingsøyr: "PMA – Never leave a project without it" IEEE Software, special issue on knowledge management in software engineering, vol. 19, pp. 43–45, 2002.
5. T. Dybå and N.B. Moe: "Rethinking the Concept of Software Process Assessment", Euro-SPI 1999, October 25–27, 1999, Series A – Pori School of Technology and Economics, No A 25.
6. B. Moltun: "A Management Concept as a Cultural Praxis", PhD. Thesis, Norwegian University of Science and Technology, May, 2004.

Empirical Evaluation of Two Requirements Prioritization Methods in Product Development Projects

Laura Lehtola and Marjo Kauppinen

Helsinki University of Technology, Software Business and Engineering Institute,
P.O. Box 9210, FIN-02015 HUT, Finland
{laura.lehtola,marjo.kauppinen}@hut.fi

Abstract. Requirements prioritization is recognized as an important but difficult activity in product development. The literature offers methods for requirements prioritization, but many authors report that practices in companies are mostly informal. In this study, we evaluated two requirements prioritization methods from the requirements engineering literature in industrial product development projects. In the first case, the users of the system evaluated the *pair-wise comparison technique* [5] for prioritizing user needs. In the second case, practitioners evaluated *Wiegers' method* [18] for change requests. The findings from the cases provide information about the suitability of the prioritization methods for product development work. In addition, our findings indicate why it might be challenging for practitioners to employ a requirements prioritization method.

1 Introduction

Requirements prioritization is defined as an activity during which the most important requirements for the system should be discovered [17]. It is recognized as an important, but, at the same time, challenging activity in requirements engineering (RE). The origins of its importance are in limited product development resources, since time and money are finite in practice. Only a limited set of requirements can be implemented in one release, but the product should, however, meet the needs of the customers and reach the markets in time [6, 16]. In practice, this means that trade-offs have to be made during the development work. For example, Wiegers [18] points out that priorities are necessary, not just so as to be able to ignore the least important requirements but also in order to resolve conflicts and plan for staged deliveries.

Challenges in requirements prioritization mainly originate from the tough nature of the issue. It is widely accepted that requirements prioritization needs complex decision-making [3, 6, 11, 13]. In order to prioritize requirements successfully, domain knowledge and estimation skills are required [8]. In practice, decision-makers find that it is not easy to define the factors on the basis of which prioritization decisions should be based, and it might be difficult to get real information that can act as a basis for the decisions [11]. In addition, requirements depend on each other and priorities are always relative. An important requirement in one release or to a certain customer may not be as important in the next release or to another customer [3]. Even the concept of prioritization is recognized as ambiguous in practice [11]. Our study showed that the term "priority" was, in some cases, used, for example, as a quantity

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 161–170, 2004.

meaning "the importance of a requirement to the customer", while in other cases it described how soon the requirement would be implemented [11].

Despite the challenging nature of the problem, there are several requirements prioritization methods introduced in the literature (for example [2, 5, 6, 18]). The high-level goal of this study is to investigate how two existing requirements prioritization methods, the pair-wise comparison technique [5] and Wiegers' method [18], suit for real product development projects. As a contribution, we provide the lessons learned for practitioners and for researchers.

This paper describes the lessons learned from one organization that tried to choose a suitable prioritization method for their product development projects by evaluating two candidate methods. The paper is structured as follows: requirements prioritization methods from the literature are introduced in Chapter 2; the empirical study is described in Chapter 3; the experience of the empirical evaluation of the methods is described in Chapter 4, and the lessons learned from the cases are explained in Chapter 5. Finally, we summarize the conclusions.

2 Requirements Prioritization Methods in the RE Literature

The requirements prioritization approaches introduced in the literature can roughly be divided into two categories: *methods* based on giving values to different factors of requirements and *negotiation approaches*. The division between these two approaches is not very strict, but the main differences are discussed in the next two paragraphs.

The *methods* are based on the idea that a priority of a requirement is a combination of the values given to the different factors affecting the priority. The methods can be divided into two subcategories. The first subcategory consists of the methods in which each requirement is processed uniquely (for example [4, 18]); the other subcategory contains the methods that are based on comparing requirements with each other (for example [5, 6]). *Negotiation approaches* encapsulate the idea that the priority of a requirement is determined by reaching agreement between the different stakeholders. Boehm's WinWin model [2] suggests identifying key stakeholders, identifying their win conditions, and negotiating about win-win reconciliations on that basis.

The origins of negotiation approaches are in contract situations, where developers and their specific customers need to negotiate about the requirements for a specific project. However, as the basic planning problems differ between traditional project-oriented systems development and market-driven product-oriented development [3, 15], we do not deal with negotiation approaches in this study.

2.1 Priority Groups, the Pair-Wise Comparison Technique, and Wiegers' Method

Priority groups, in which requirements are put into categories, is the most traditional and best-known requirements prioritization practice. For example, IEEE [4] recommends ranking requirements according to their importance using scaling: Essential, Conditional, and Optional, referring to the requirement's contribution to the acceptability of the software. Kovich [9] introduces a slightly different approach in which requirements are categorized on the basis of how perfectly they must be implemented.

Karlsson's *pair-wise comparison technique* [5] is based on the *Analytical Hierarchy Process (AHP)* [14]. In AHP, all possible pairs of hierarchically classified attributes are compared pair-wise, according to how well they contribute to the attributes in the upper hierarchy level. In the pair-wise comparison technique, all requirement pairs are compared according to their importance. The scale used in the comparisons is basically the same as that employed in AHP: 1 (of equal importance); 3 (moderate difference in importance); 5 (essential difference in importance); 7 (major difference in importance), and 9 (extreme difference in importance). By means of the calculations introduced in [5], these comparisons lead to an understanding of what each requirement's share of the total value of the requirements is. In [6], Karlsson and Ryan introduce the *cost-value approach*, which improves the pair-wise comparison technique by using cost and value as high-level factors against which each requirement pair is compared.

Wiegers' method [18] is based on the idea that the priority of a requirement can be calculated by dividing the value of a requirement by the sum of the costs and technical risks associated with its implementation. The value of a requirement is seen as being dependent both on the value the requirement provides to the customer(s) and the penalty incurred if the requirement is absent. According to Wiegers [18], developers should evaluate the requirement's costs and implementation risks, as well as the penalty the developers may incur in customers' eyes if the feature is missing. The customers should evaluate the value the requirement provides for them. Each attribute is evaluated on a scale from 1 to 9.

Table 1. Three existing requirements prioritization methods from the evaluator's viewpoint

Method	What the evaluator must do	Scale
Priority groups	Divide requirements into three categories according to a criterion (usually "importance").	• For example: Essential, Conditional, Optional
Pair-wise comparison technique	Compare each of the possible requirement pairs and decide how much more valuable the other requirement is.	• 1, 3, 5, 7, 9
Wiegers' method	Evaluate each requirement according to its value to customer, penalty if it isn't implemented, implementation costs, and risks.	• 1-9

3 Research Methods

The experience drawn on in this research comes from work with an industrial partner of the Qure[1] research project at Helsinki University of Technology. The high-level research goal of the Qure project was to investigate how organizations can develop products that better satisfy user and customer needs. This research activity was part of

[1] The Qure project was a research project funded by the National Technology Agency of Finland (Tekes) and industrial partners.

a subproject investigating how user and customer needs and requirements should be prioritized.

The objectives of our research were to characterize the benefits and challenges of existing requirements prioritization methods in product development work. In order to clarify this area, we evaluated two prioritization methods, the pair-wise comparison technique [5] and Wiegers' method [18], in industrial product development projects.

3.1 Case Company

The research work was carried out in the R&D unit of a Finnish company developing transportation systems for buildings. At the time of the study, there were 23,000 employees in the company. The case organization had invested a substantial amount of resources in RE process improvement. At the time of this study, there was RE improvement project in progress. The idea of improving requirements prioritization in the organization was one improvement action taken by the RE process improvement group.

3.2 Research Steps

In order to improve the prioritization practices in the organization, an improvement group was established in the case organization. The goals for this temporary group were to find out a suitable requirements prioritization method from the literature, evaluate it, and introduce and adopt it in the product development organization. The group consisted of a usability expert, a visual designer, two project managers, and an external researcher who worked as the facilitator of the group.

At the time of the study, the authors were active participants in the RE process improvement work in the case organization. This made it possible for us to use an action research approach and, via that, to have a deeper insight into what the practitioners do, not just what they say they do [1]. The practical role of the researcher in the improvement process was to function as a kind of a facilitator or moderator. She interviewed the usability expert before the improvement group was established, in order to have a wider understanding of the situation in the organization at that time. In addition, she introduced the prioritization methods to the improvement group, gathered and documented the experience, presented the findings from the cases to the group members, and supported the improvement work.

3.3 Data Collection

We evaluated two existing requirements prioritization methods, the pair-wise comparison technique [5] and Wiegers' method [18], in real development projects. Both of the studies were performed in separate projects, which presented different challenges. The two case projects were selected on the basis of their interest in adopting a prioritization method. Both of the projects decided to try the prioritization method that they thought would be the most suitable in their case.

In both cases, we selected relevant participants within or outside the project and a subset of the project's requirements. The participants prioritized the requirements with the prioritization method according to the instructions given in the literature.

However, the project managers were given the opportunity to make small adjustments to the methods if they felt that something in the method would not be suitable in their case. The cases are briefly summarized in Table 2.

Table 2. Case projects

	Type of requirements	Users	Prioritization method
Project A	User needs	4 users of the system	Pair-wise comparison technique
Project B	Change requests	Project manager Requirements engineer	Wiegers' method

The role of the researcher was to provide help in the usage of the methods and to collect the experiences by making notes and having discussions with the participants. The researcher also interviewed the practitioners about the usage of the method and how they felt about the prioritization results that the methods gave. In addition, the researcher helped in calculating the prioritization results according to the data collected and presented the findings to other improvement group members.

Case 1: Pair-Wise Comparisons for User Requirements

Project A was an investigation project during which the organization wanted to have more information about real user preferences concerning one of the main products of the company. At the time of the study, the practitioners involved had gathered a good deal of data concerning user needs by performing a field study (introduced in [10]) and they were satisfied with their manifold new findings. However, all the user needs had an equal value in the report and the practitioners did not know how to select the most important needs for further recognition. They wanted to introduce a common prioritization method to the project in order to systematically prioritize the user needs.

We invited four users of the product to prioritize the gathered user needs by means of the pair-wise comparison technique [5], while the project manager wanted to know which of the requirements are the most important to the users. We grouped the requirements into ten categories of 20 or less requirements. The categories were such as "Usability" or "Performance". We predetermined all of the users to prioritize the same four categories of requirements. However, we found with the first user that it was impossible to prioritize the biggest category. That is why we left the biggest category out with the next three users, but one new category in.

Case 2: Wiegers' Method for Change Requests

In Project B, there were numerous change requests that had been gathered by a project manager, but no ideas how to systematically prioritize these. The project manager had tried to prioritize the requirements mutually, but felt that he needed a systematic way to find which of the requests to implement first.

In this case, both the project manager and another project member prioritized 6 change requests using Wiegers' method [18]. In this case, the practitioners wanted to use the method as it is described in the literature. However, as they faced some challenges concerning the usage of the method during the prioritization, the product manager made his own adjustment to it. This is discussed in more detail in the next chapter.

4 Experience from the Empirical Evaluation of the Methods

The next two sections summarize the findings from the two cases. We especially point out the challenges involved in order to gain a better understanding about the limitations of the prioritization methods in real product development projects.

4.1 Pair-Wise Comparison Technique

In the pair-wise comparison case, the selected users were interested in sharing their preferences concerning the product with the product developers and took the practitioners' interest in their opinions as an honor. The product developers, on the other hand, found the idea of getting a list of requirements, put in descending order according to their importance to users, attractive. From the methodological point of view, the users found it quite easy to say which of the two requirements was more important in each case. However, we found some challenges in the usage of the method, which we describe in the following paragraphs.

Users found it difficult to estimate how much more valuable one requirement is than another. Even though the users found it easy to decide the order of importance of each pair of requirements, they had difficulties in nominating the extent to which one is more important than the other. Karlsson [5] recommends using an importance scale of up to 9 steps, but even 5 steps was too much for the users in our case. In addition to the verbally announced and observed difficulties, none of our users used more than 3 importance steps (for example 3, 5, and 9) with one set of requirements.

Pair-wise comparisons with over 20 requirements were difficult in practice. After working for half an hour with the same 20 requirements, our first user become so irritated that she was not really concentrating on comparing the requirements. All she wanted was just to finish the task fast. We had to change the research setting and abandon this particular category of requirements with the other three users.

Requirements at different levels of abstraction caused trouble. According to the preliminary nature of user requirements, the requirements in our comparison were at quite different levels of abstraction from each other. Even though we had grouped the requirements, there were, for example, requirements such as "The materials should be of good quality" and "Stone should be one material alternative" in the same group. "Stone" might be one component of quality, but in this case the users preferred the more general "quality". The prioritization results we got easily gave the illusion that stone is not a good material alternative at all, but there were just no other material alternatives in this comparison.

Some users conceived pair-wise comparisons as pointless. Two of the users complained that the pair-wise comparisons were pointless. They felt that it would have been easier for them just to select the most important requirements or put the requirements in descending order without any pair-wise comparisons.

4.2 Wiegers' Method

In the Wiegers' method case, the practitioners were enthusiastic about the idea of getting their manifold change request distributed to a continuum as an outcome. In

addition, they found the idea of evaluating the requirements in a controlled way from different viewpoints interesting. However, our findings concerning the method are not only positive.

It was unclear for practitioners on what information they should base the evaluations of the factors. It was unclear for practitioners what the real-life elements of the value, penalty, cost or risk in their case actually were. "How do I evaluate a requirement's value to the customer or implementation costs if we do not have any common basis as to what the terms "value" or "cost" mean? What is the information on which I should base my evaluation?" complained one practitioner.

Practitioners found it difficult to estimate which number to give to factors. It was not easy for the practitioners to decide what number to give in each case to the factors. "What does it mean if I give either number 3 or 5? What is the practical difference between them?" argued one practitioner. Both of the practitioners felt that they were missing common guidelines regarding the meanings of different scale levels in their case.

Practitioners created their own formulas in order to be able to evaluate the factors. In order to evaluate "value" or "risk" and attribute numbers to these factors, the project manager decided to create his own mathematical formulas. On one hand, this led to better understanding of the information that the evaluation should be based on in their case, but on the other hand it also led to complicated and troublesome calculations without mathematical relevance.

Practitioners changed their estimates if the order of requirements emerged "wrong". Before the prioritization, the practitioners said that they were unsure about the priority order of their requirements and were therefore enthusiastic about using prioritization methods. However, we found that the practitioners had strong opinions about the priorities and that in some cases they tried to use the prioritization method in order to prove their existing views. For example, if a requirement they had preferred got low priority according to the formulas, they changed their estimates so that the preferred requirement became more important in the priority list.

5 Lessons Learned

The lessons learned from evaluating existing requirements prioritization methods in the two case projects are summarized in bullet points below and described in Sections 5.1-5.4:

- Practitioners seem to mistrust the results they get with the methods
- A lack of clarity concerning the usage of the prioritization methods may affect the prioritization results
- Prioritization results may lead to a wrong impression of what to do on that basis
- Practitioners need project-specific guidelines

5.1 Practitioners Seem to Mistrust the Results They Get with the Methods

Practitioners seem to mistrust the results they get by using prioritization methods. In our cases, the practitioners were interested in using the methods and felt that they would need one to make better prioritization decisions. However, if the priority order

given by the method was contrary to their experience-based opinions, they felt that the method was not working properly.

The mistrust of the methods was expressed differently in the two cases. In the Wiegers' method case, practitioners changed their estimates in order to get a "better" priority order if the results given by the method seemed wrong. In the pair-wise comparison case, the users argued that pair-wise comparisons are gratuitous and that they could just select the most important requirements without comparisons.

5.2 A Lack of Clarity Concerning the Usage of the Prioritization Methods May Affect the Prioritization Results

Aspects of the usage of the prioritization methods that are unclear may lead to wrong calculations and thereby be incorporated into prioritization results. In our cases, there occurred some practical problems during the prioritization work. We found that these affected the priority order of the requirements.

The prioritization results are never better than the raw data inserted. If a user evaluates the factors in a hurry without careful consideration or cannot nominate the extent to which one requirement is more important than the other, or if a practitioner does not know what a requirement's real value for the customer is, the prioritization results are nothing more than just rough guesses. Getting exact numbers or fractals as an outcome does not ensure the validity of the results, since the mathematical calculations cannot improve the quality of the raw data inserted.

5.3 Priority Results May Lead to a Wrong Impression of What to Do on That Basis

Requirements prioritization methods give a priority list as an outcome. These lists may, in some cases, lead to a false impression among the practitioners as to what one should do on the basis of them. In our case, the practitioners found that it was not possible to just select the first requirements from the priority list and be sure that these are the most important requirements that should be implemented first.

In the pair-wise comparison case, it seemed that some requirements were getting low priorities just because they were not compared to the other requirements of the same level of abstraction. For example, in one requirements group, a more general "quality" was prioritized over a specific material alternative when they were compared. However, it would not be right to indicate on that basis that this material alternative could not lead to quality or be an excellent material alternative. In the Wiegers' method case, we also found that taking value per cost plus risk ratio may lead to a priority list that favours unremarkable requirements with both low value and costs and undervalues important but expensive requirements (for example: 2/1 is a bigger number than 8/7).

5.4 Practitioners Need Project-Specific Guidelines

If an organization wants to use a prioritization method, practitioners need commonly defined guidelines about the elements of the factors and usage of the scales. Practitioners need to define, for example, what viewpoints should be taken into account when evaluating the "value" and "cost". In addition, they need to negotiate what the difference between, for example, the numbers 3 and 5 on a scale means in practice.

In our cases the users of the prioritization methods lacked a common understanding of the rules concerning the usage of the methods. This led to a situation in which all the participants had their own understanding of the basis on which the factors should be evaluated. In addition, every user used the scales differently, which made it difficult for them to combine their points of view.

6 Conclusion

In this paper, we evaluated two requirements prioritization methods, the pair-wise comparison technique [5] and Wiegers' method [18], in two industrial case projects. On the basis of the study, we provide information regarding how these methods suit real product development projects and discuss issues that practitioners should recognize when using them.

The case organization of this study wanted to improve their informal requirements prioritization practices by employing a systematic method. The lack of, and demand for, systematic requirements prioritization practices in companies is recognized in the literature as well [12, 6, 8]. Our study indicates that the evaluated requirements prioritization methods provide help for practitioners. However, requirements prioritization seems to be such a complex decision-making problem that organizations cannot solve it comprehensively by just employing one method.

According to our study, one possible benefit of using requirements prioritization methods is that practitioners acquire a more controlled way of taking different viewpoints into account during prioritization. Even rough considerations of, for example, user preferences and implementation costs may provide help in decision-making. It seems that the benefits of using a systematic prioritization method are not just getting a priority list as an outcome. Similarly, Karlsson et al. [7] discuss positive side effects occurring during their pair-wise comparison sessions, such as identifying ambiguous requirements.

However, the practitioners in both of our cases faced practical difficulties with the methods. For example, in order to prioritize requirements with the methods, the practitioner has to evaluate factors such as "value" or "cost". In our cases, it was not clear for practitioners on what information they should base their evaluations of these factors. According to our findings, we would recommend practitioners to agree before prioritization on what information the evaluations should be based on and how to use the scales. For example, project members could agree that they should consider issues such as importance for users, importance for customers, and importance of the requirement's original source when evaluating the "value" of a requirement.

One challenge in our cases was that the practitioners seemed to mistrust the results they got by the methods. The experiments conducted by Lena Karlsson et al. [8] reveal somewhat similar findings. Some of the evaluators of the requirements, in their case, felt a loss of control over the prioritization process when they used the analytical hierarchy process (AHP).

In summary, our study indicates that prioritization methods can provide help in putting a set of requirements in order and the results of the prioritization may work as a basis for discussion. However, there are practical difficulties in the usage of methods and therefore prioritization results should be taken more as being indicative than as an ultimate truth. Because of the small sample of case projects and limited set of prioritized requirements, the findings of our study may not be valid in all organiza-

tions. However, our study provides directions for further investigation. One of the research challenges in the future is to investigate if project-specific agreement on the information that evaluation of the factors should be based on improves the use of prioritization methods and affects the practitioners' trust of the prioritization results.

References

1. Argysis, C., Putman, R., Smith, D.: Action Science. Jossey-Bass, San Francisco (1985)
2. Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., Madachy, R.: Using the WinWin Spiral Model: A Case Study. IEEE Computer 31 (7) (1998) 33 – 44
3. Carlshamre, P.: Release Planning in Market-driven Software Product Development - Provoking an Understanding. Requirements Engineering Journal 7 (3) (2002) 139-151
4. IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications (1998)
5. Karlsson, J.: Software Requirements Prioritizing. In: Proceedings of the 2nd IEEE International Conference on Requirements Engineering (ICRE1996), Colorado, USA (1996) 110-116
6. Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software 14 (5) (1997) 67–74
7. Karlsson, J., Wohlin, C., Regnell, B.: An Evaluation of Methods for Prioritizing Software Requirements. Information and Software Technology 39 (14-15) (1998) 939–947
8. Karlsson, L., Berander, P., Regnell, B., Wohlin, C.: Requirements Prioritisation: An Experiment on Exhaustive Pair-Wise Comparisons versus Planning Game Partitioning. In: Proceedings of Empirical Assessment in Software Engineering (EASE2004), Edinbourgh, Scotland (2004)
9. Kovitz, B.: Practical Software Requirements: A Manual of Content and Style. Manning Publications Co, Greenwich (1999)
10. Kujala, M., Mäntylä, M.: Studying Users for Developing Usable and Useful Products. In: Proceedings of 1st Nordic conference on Computer-Human Interaction, Stockholm, Sweden (2000) 1-11
11. Lehtola, L., Kauppinen, M., Kujala, S.: Requirements Prioritization Challenges in Practice. In: Proceedings of 5th International Conference on Product Focused Software Process Improvement, Kansai Science City, Japan (2004) 497-508
12. Lubars, M., Potts, C., Richter, C.: A Review of the State of the Practice in Requirements Modelling. In: Proceedings of First IEEE Symposium on Requirements Engineering (RE´93), San Diego, California, USA (1993) 2-14
13. Moisiadis, F.: The Fundamentals of Prioritising Requirements. In: (Web) Proceedings of Systems Engineering/Test and Evaluation conference (SETE2002). (2002) http://www.seecforum.unisa.edu.au/Sete2002/ProceedingsDocs/.
14. Saaty, T. L.: The Analytical Hierarchy Process, McGraw-Hill, (1980)
15. Sawyer P.: Packaged software: Challenges for RE. In: Proceedings of the Fifth International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ 2000), Stockholm, Sweden (2000)
16. Siddiqi, J., Shekaran, M.: Requirements Engineering: The Emerging Wisdom. IEEE Software 13 (2) (1996) 15–19
17. Sommerville, I.: Software Engineering. 5 ed. Addison-Wesley, Wokingham, England (1996)
18. Wiegers, K.E.: Software Requirements. Microsoft Press, Redmont, Washington (1999)

Project Effort Estimation: Or, When Size Makes a Difference

Oddur Benediktsson[1] and Darren Dalcher[2]

[1] University of Iceland, Hjarðarhaga 2-4,
107 Reykjavik, Iceland
oddur@hi.is

[2] Middlesex University, Trent Park
Bramley Road, London N14 4YZ, UK
d.dalcher@mdx.ac.uk

Abstract. The motivation for this work is derived from the current interest in speeding up development schedules. A key implication of the shift to more rapid development methods is the growing emphasis on fixed time and fixed effort delivered during such projects. However there appears to be little work that addresses the impacts of dealing with bound effort levels. The result of binding time and effort is to deprive project managers of the normal parameters that are used in tradeoffs. The paper attempts to introduce a quantitative analytical framework for modeling effort-boxed development in order to uncover the effects on the overall development effort and the potential leverage that can be derived from incremental delivery in such projects. Models that predict product size as an exponential function of the development effort are used in the paper to explore the relationships between effort and the number of increments, thereby providing new insights into the economic impact of incremental approaches to effort-boxed software projects.

1 Introduction

The popular computing literature is awash with stories of IS development failures and their adverse impacts on individuals, organisations and societal infrastructure. Indeed, contemporary software development practice is regularly characterised by runaway projects, late delivery, exceeded budgets, reduced functionality and questionable quality that often translate into cancellations, reduced scope and significant re-work cycles [8]. Failures in particular, tell a potentially grim tale. In 1995, 31.1% of US software projects were cancelled, while 52.7% were completed late, over budget (cost 189% of their original budget), and lacked essential functionality. Only 16.2% of projects were completed on time and within budget; only 9% in larger companies, where completed projects had an average of 42% of desired functionality [16]. The 1996 cancellation figure rose to 40% [17].

The cost of failed US projects in 1995 was $81 billion, in addition cost overruns contributed an additional $59 billion (so that out of the $250 billion spent on 175,000 US software projects, $140 billion was spent on cancelled or over budget activities)

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 171–183, 2004.

[16]. In fact, Jones contended that the average US cancelled project was a year late having consumed 200 percent of its expected budget at the point of cancellation [10]. In 1996, failed projects alone totalled an estimated $100 billion [12]. In 1998, 28% of projects were still failing at a cost of $75 billion, while in 2000, 65,000 of US projects were reported to be failing [18].

The Standish Group makes a distinction between failed projects and challenged projects. Failed projects are cancelled before completion, never implemented or scrapped following installation. Challenged projects are completed and approved projects which are over-budget, late and fewer features and functions than initially specified. Most organisations are constantly searching for ways to improve their development practice and reduce the likelihood of failures and the degree to which their projects are challenged.

Typical projects entail a balancing act between the triple constraints of budget, schedule and scope. Trade-offs and adjustments are therefore made by restricting, adding to or adjusting the cost, time and scope associated with a project. Budding IT project managers are introduced to the holy trinity of project constraints at an early stage and are advised to balance the factors in order to define an acceptable compromise.

Indeed the traditional triangle in project management is said to be concerned with finding the balance between cost, time and scope (see Figure 1). For example, the more that is requested in terms of scope (or arguably even the performance or the quality) the more it is likely to cost and the longer the expected duration. If the client needs to have a certain performance delivered very rapidly, this will increase the cost due to the need to work faster and have more people involved in the development. The more features expected from a system, the higher the cost and the longer the expected duration. If the costs need to kept to a minimum, one may need to consider the essential performance, or the overall scope, and compromise there.

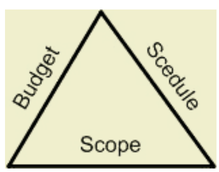

Fig. 1. Budget, Schedule and Scope trade off

The three factors are obviously closely entwined and project managers are expected to balance the **what** (scope) with the **when** (schedule) and the **how much** (budget). The trade-off between time, scope/performance and cost is essential in understanding the tension in a typical project management environment.

Many managers quickly discover that the triangle is not flexible. Performance and scope are often pre-determined prior to the project. Moreover, project managers often inherit the overall budget from the contracting activities that may even have imposed a fixed-price contract structure. A fixed overall budget may also exclude the hiring of specialists and the addition of human resources. The only remaining scope for leverage is in the schedule. However, this may also be imposed on a project through a

fixed date for delivery with little regard to the complexity of the intended system or the risks it embodies. Once both budget and schedule are fixed, there appears to be little scope for compromises and tradeoffs.

The three factors play a key part in determining the degree to which a project is challenged (or even deemed a failure), yet they may be uncontrollable by the project manager. Indeed, Capers Jones observed that the most common constraints encountered are: fixed delivery dates; fixed-price contracts; staffing or team size limitations; and performance or throughput constraints [11] i.e. fixed time, price, staffing level (see Section 2), performance and scope. Many managers are thus looking to control other factors that may alter the outcome of the project, especially as the constraints often occur in concert. An alternative solution is to opt for iterative or incremental delivery of partial, yet useful versions of the system. Such an approach can rely on the success of earlier deliverables and build on them to upgrade the system. This can be achieved within the overall timescale whilst adhering to the intended scope of the system. The approach is useful in trying to mitigate the effect of failures by reassessing the outcome after achieving each increment or deliverable milestone. It also empowers project managers by enabling them to operate and make decisions within the traditional triangle of project management.

2 Working Incrementally

In many modern IT projects, the desire to avoid failures is further exacerbated by the need to deliver almost instantaneous functionality in response to changes. The pressure to collapse the schedule is typically due to the need to respond to opportunities, time-to-market demands, the need to get there before the competition or the implications of technical obsolescence. As organisations race to ensure a functioning presence on the web, the new mode of operation is introducing additional constraints forcing new working patterns. The market is fast-moving characterised by a dynamic environment with high levels of uncertainty and risks. Customers appear more demanding and discerning expecting non-stop service around the clock. Service is being judged according to overall traffic, frequency and duration of visits and loyalty.

The rate of change endemic to this environment leads to small and quick projects with high frequency of release and smaller scope explored in each release (akin to timeboxing). The scope is highly changeable over time to respond to competition and opportunities. Time therefore becomes the critical factor: As the speed of release is measured by getting there before the competition, project execution is measured according to the shortest time to register a presence (often regardless of the quality). Trade-offs can often sacrifice scope, cost, expectations or quality to accelerate the speed of completion.

Most projects in this environment are new, innovative and difficult to estimate and cost. The planning approach thus needs to focus on key milestones and targets, yet remain flexible enough and responsive enough to cope with changing requirements, delivery dates, release deadlines and new opportunities. Projects tend to be evolutionary and have short release cycles as they build on what already exists and are often prone to scope re-adjustments in line with shifting knowledge and expectations.

Incremental and iterative development approaches have long been recognised as effective in reducing the risk of failure in such situations as they entail a more controlled approach to development [2]. Incremental approaches emphasise phased development by offering a series of linked mini-projects (referred to as increments, releases or versions) [9]. Work on different parts and phases, is allowed to overlap throughout the use of multiple mini-cycles running in parallel. Each mini-cycle adds additional functionality and capability. The approach is underpinned by the assumption that it is possible to isolate meaningful subsets that can be developed, tested and implemented independently. Delivery of increments is staggered as calendar time progresses. The staggered release philosophy allows for learning and feedback to alter some of the customer requirements in subsequent versions. Incremental approaches are particularly useful when there is an inability to fully specify the required product or to fully formulate the set of expectations under some budgetary control.

Note that the term 'Incremental development' as used here embraces the spectrum of iterative development and delivery (ranging from the full product architecting and design being performed upfront, to evolutionary methods and may also be extended to refer to delivery modes in agile methods with little or no a priori product design).

Indeed in the context of managing projects within the triangle of constraints, the adoption of **incremental development**, **Rapid Application Development or** the use of **agile methods** enables project managers to gain flexibility within the pre-defined boundaries of overall time and cost. These approaches are focused on the need to deliver relevant working business applications quicker and more cheaply. The application is likely to entail delivery in incremental fashion while maintaining user involvement through the application of design teams and special workshops. Projects tend to be small and limited to short delivery periods to ensure rapid completion and are thus particularly suitable as controlled cycles of small implementation released as part of a larger project. The management strategy utilised in incremental development relies on the imposition of *timeboxing*, the strict delivery to target which dictates the scoping, the selection of functionality to be delivered and the adjustments required to meet the deadlines. Rapid Application Development and agile methods are particularly useful in environments that change regularly and impose demands of early solutions.

The motivation for this work is thus derived from the current interest in speeding up development schedules. A key implication of the shift to more rapid development methods is the growing emphasis on fixed time and fixed effort in projects. Going back to the triple constraint, if time schedules are fixed by timeboxes and costs are largely dictated by the availability of personnel to work within these imposed time frames, the main variable is the scope that can be delivered. However, the scope (including the actual delivered and useful functionality and the quality or usability of that functionality) is the driving concern in many projects. Indeed, scope needs to be handled with care as delivering partial functionality (e.g. the release of a sophisticated order recording systems on the web a few weeks prior to Christmas without a corresponding order fulfilment system capable of despatching the ordered goods).

Project managers respond to such constraints by trying to adjust the other parameters of the project. However, assuming that overall cost, time and scope are fixed,

which variable remains to be manipulated? Fixed time boxes in fixed time frames, result in *chunks of fixed person-months* required to complete the task. The key question is, given the reality of fixed chunks of person-months, are there any methods for manipulating person-months in an attempt to address project objectives and leverage off different arrangements? More crucially perhaps, we are interested in finding out how much can be accomplished with such a given effort.

The current literature offers very little clues about what can be accomplished. The Standish Group recommended that an ideal (and achievable) project would entail six developers working for six months [16]. This confirms the notion of fixed developer resources imposed on the project. These results are corroborated by a large study of success and failure in over 1,000 UK IT projects which found that 90% of the successful projects were less than 12 months long and 47% had a duration of less than 6 months [20]. Note however, that deploying a fixed team of six developers for six months ultimately gives 36 *PM* as a fixed chunk of key project resources that can be manipulated to ensure that the project does not get out of hand. Recently, the Standish Group revised the figures to suggest an ideal team of four developers working for four months, giving a much reduced total of 16 *PM*. (Note that a person-month is defined in the computing literature as the amount of time one person spends working on a software development project for one month.) Indeed, the Standish conclusions assert that 'shorter timeframes with delivery of software components early and often, increase the success rate' [18]. Moreover, they implicitly confirm that managers now have a new type of resource to manipulate.

The interest in agile methods and smaller and rapid projects is driving down the average duration and fixed level of resources and hence the *project resource envelope* that can be utilised by project managers. Indeed, in Extreme Programming developers often talk about even smaller chunks of development (measured in weeks and therefore totalling 1-4 months) while DSDM recommends timeboxes of three month duration [19]. The figures take us back to the all important question: 'Does size matter?'

Earlier work attempted to explore the relationship between effort and the number of increments [2]. A key conclusion of that work was the realisation that the ratio of work (and hence the overall work) decreases as the number of increments increases. Indeed, using multiple increments appears to offer great benefit compared to monolithic development approaches. This work can now be extended to explore the impact of dealing with fixed time, scope and effort parameters by dividing the fixed *project resource envelope* authorised for a given project into a set of increments.

The rest of the paper will focus on providing a framework for evaluating the relationship between project size and the effort required to deliver it in the context of a fixed effort level. The framework will then be populated with a range of values to provide a sampling of scenarios for different project environments. These scenarios offer an insight into the behaviour patterns expected under different conditions. The paper concludes by drawing a set of observations from the results and commenting about the implications of the results and about the ideal size and complexity relationship, before finally re-assessing the issue of ideal size in order to provide practical answers.

3 Size of the Development Effort

Estimating the development effort for a software system is a long standing problem in software project management. It has generally been noted that the effort (*E*) is strongly correlated to the program size. A great deal of research has been carried out to relate the effort empirically to the product size, with the effort being commonly expressed in Project Months (PM) and the size in Kilo Lines of Code (KLOC) or in Functions Points (FP). Typical models used to express this relationship are of the form:

$$PM = c + a\, Size^{\,b} \tag{1}$$

where a, b, and c in equation (1) are determined by regression analysis using a collection of project outcomes. Some of the better known empirical models are highlighted in Table 1 [3, 4, 14, 21].

Table 1. Effort estimation models

Author	Original formula	Inverted form
Halstead	$PM = 0.7\ KLOC^{1.50}$	$KLOC = 1.27\ PM^{0.667}$
Basic COCOMO organic	$PM = 2.4\ KLOC^{1.05}$	$KLOC = 0.43\ PM^{0.952}$
Basic COCOMO semidet.	$PM = 3.0\ KLOC^{1.12}$	$KLOC = 0.37\ PM^{0.893}$
Basic COCOMO emb.	$PM = 3.6\ KLOC^{1.20}$	$KLOC = 0.34\ PM^{0.833}$
COCOMO II.2000	$PM = 2.9\ KLOC^{1.10}$	$KLOC = 0.38\ PM^{0.909}$
Walston-Felix	$PM = 5.2\ KLOC^{0.91}$	$KLOC = 0.16\ PM^{1.10}$
Bailey-Basil	$PM = 5.5\ KLOC^{1.16}$	$KLOC = 0.23\ PM^{0.862}$
Doty (for KLOC > 9)	$PM = 5.288\ KLOC^{1.047}$	$KLOC = 1.27\ PM^{0.674}$
Albrecht and Gaffney	$PM = -13.39 + 0.0545\ FP$	$FP = 245.7 + 18.35\ PM$
Kemerer	$PM = 60.62 \times 7.728 \times 10^{-8}\ FP^{3}$	$FP = 59.76\ PM^{0.33}$

Column three of Table 1 is derived from column two by solving for *Size* in terms of *PM*. The models are seen to vary considerably and can be expected to give different results in individual estimation cases. The estimation accuracy is improved by setting a number of parameters to reflect the situation at hand regarding the product, process, platform, and people involved as is done for example in COCOMO II [4].

The terminology used in this paper is that of the COCOMO II model [4]. COCOMO II represents an update to the original COCOMO 81 work [3] to account for contemporary development practices. The paper builds on earlier work [2, 6] that utilises the notation as a quantitative analytic framework for evaluating software technologies, project decisions and the economic impacts in terms of the resulting projects. The focus in this work is on relating the *Size* of the outcome of a development project to the input effort. Column three of Table 1 gives expressions for this relationship. Note that the *Size* (KLOC or FP) is an exponential function of *PM* with the exponent being less than or equal to one in all but the Walston-Felix model. The relation that will be exploited here is of the form

$$Size = c\, PM^{\,d} \tag{2}$$

which is in line with all of the above mentioned models except the Albrecht and Gaffney model which has a constant term.

4 Size in Incremental Development

Suppose that *PM,* the total allocated project effort, is fixed for a given development project (e.g. following the recommendation of the Standish Group or as proposed by the proponents of agile methods and DSDM) and that the nature of the project is such that it can be broken into a sequence of *n* increments. The effort exerted in the individual increments is denoted by PM_1, $PM_{2, \ldots}$,PM_n where

$$PM = \sum_{i=1}^{n} PM_i \tag{3}$$

While incremental development aims towards a target functionality, there has to be some additional effort used for the integration of the different increments. The term used for this in the COCOMO models is *breakage* as some of the existing code and design has to be mended to fit a new increment in. The breakage effect will be accounted for by reckoning that some fraction of $PM_{i,}$, the incremental effort exerted in increment *i*, is gained as system enhancement. This fraction is termed *Gain Multiplier (GM).* The net effort is expressed as $GM_i \, x \, PM_i$.

Kan asserted that 20% of the added code in staged and incremental releases of a product goes into changing the previous code [13]. Cusumano and Selby reported that features may change by over 30% as a direct result of learning during a single iteration [7]. In a recent paper the authors argued that the incremental integration *breakage* can be expected to lie in a range from 5% to 30%. This relates to the range in Gain Multipliers of 95% down to 70% in approximate terms [2]. Thus a Gain Multiplier can assume a range of values. The GM will equal *1* when all of the gross work delivers new functionality. Where some gluing, integration or refactoring effort is needed the efficiency will be reduced below *1*. The gain factor can exceed *1* in the case of marked code reuse (thus suggesting negative entropy and a major benefit from the application of reuse). One might ask if the Gain Multiplier can be negative in which case adding an increment is counterproductive – i.e. more code needs to be mended than has been written. This interesting pathological case is outside the scope of this work. It is therefore assumed here that $GM>0$.

The net incremental effort exerted for increment *i* is $GM_i \, x \, PM_i$. The $Size_i$ of a developed increment is then derived from expression (2)

$$Size_i = c_i \, (GM_i \, PM_i)^{d_i} \tag{4}$$

This relation does not hold if the increments are “too small”. For example, the COCOMO formulation stipulates that the size has to be larger than 2 KLOC (which roughly equates with 6 PM). Note also that for one-off development $n=1$ and $GM=1$ which is consistent with expression (2).

When the outcome of the individual increments is accumulated the total developed size $Size_I$ is

$$Size_I = \sum_i Size_i \tag{5}$$

and by combining (4) and (5)

$$Size_I = \sum_i c_i\,(GM_i\, PM_i)^{d_i} \tag{6}$$

When the development context is uniform (i.e. one and the same team throughout as well as homogeneous project and process properties) then the c and d parameters in (6) are taken to have constant values across the increments i.e.

$$c_i = c \ \text{ and } \ d_i = d \quad \text{for} \quad i = 1,\ldots n$$

and then expression (6) simplifies to

$$Size_I = c\sum_i (GM_i\, PM_i)^d \tag{7}$$

Expressions (6) and (7) are seen to represent **a general way of estimating the outcome of incremental development projects.**

The following question arises: Does the number of increments matter for the resulting product size? Expression (7) will be explored below in a search for an answer to this question.

First, the one-off case (expression (2)) is employed to normalize expression (7) resulting in what is termed here the *Relative Productivity (RLPR)* that is $RLPR = Size_I /(c\,PM^d)$. By combining (2) and (7)

$$RLPR = \sum_i (GM_i\, PM_i / PM)^d \tag{8}$$

The multiplier c in (2) and (7) has been factored out in the normalization. *RLPR* is seen to give clear indications of productivity in incremental development - for $RLPR > 1$ the productivity has increased while for $RLPR < 1$ the productivity has decreased as compared to the one-off case.

Second, suppose that the total effort exerted in the n increments is equally divided. In other words, the work is effort-boxed. In this case $PM_i = PM / n$ for all i. Then expression (8) simplifies to

$$RLPR = n^{-d} \sum_i (GM_i)^d \tag{9}$$

As mentioned earlier effort-boxed development is gaining in popularity. Evidence of this trend can be seen as a result of wide adoption and interest in agile methods and extreme programming as well as the use of the effort-box notion in IT projects. Moreover, improved failure statistics in failure surveys is directly attributable to the increasing proliferation of mini projects.

Third, a parameter termed the *Cumulative Gain (CG)* is now defined as

$$CG = 1/n \sum_i (GM_i)^d \tag{10}$$

CG is a function of n. For $d=1$ (the linear case) *CG* gives the arithmetic average of the GM_i values. For a constant *GM* across the increments denoted as $\underline{GM}$ then $CG=\underline{GM}^d$. In general CG can be expected to be a complex function of a) the devel-

opment context of a project, b) the number of increments, and c) the extent of reuse. Expressions (9) and (10) combined yield the following compact expression for the Relative Productivity:

$$RLPR = n^{1-d} CG \tag{11}$$

The parameter d is taken to be restricted to $0<d<1$ (see Table 1 Column 3). In the case where CG is constant, then expression (11) is seen to be a monotonic increasing function of n i.e. the productivity increases with increasing number of increments.

Note that the underlying assumptions in this expression (11) are a) effort-boxed increments and b) uniform development context. This expression will be exploited in the next section in the quest for gaining additional insight into the behaviour of the incremental development model.

5 Discussion

Table 2 shows sample computations of *RLPR* using expression (11). Relative Productivity values that are less than 1 are italicised. For these values there is no gain in using incremental development in terms of reduced development effort. Below the italicised region, *RLPR > 1* and the productivity increases as compared to the one-off development.

Table 2. Relative Productivity ***RLPR*** for the exponent fixed at ***d = 0.833***

	CG Cumulative Gain			
n	**1**	**0.9**	**0.8**	**0.7**
1	1.00	*0.90*	*0.80*	*0.70*
2	1.12	1.01	*0.90*	*0.79*
4	1.26	1.13	1.01	*0.88*
6	1.35	1.21	1.08	*0.94*
8	1.42	1.27	1.13	*0.99*
10	1.47	1.32	1.18	1.03
12	1.51	1.36	1.21	1.06

The productivity is seen to increase down a column for constant Cumulative Gain as has been observed. However, for a particular project, a planning scenario could possibly yield the incremental alternatives presented in Table 3. The planning alternatives that could be utilised by the project manager are the *(n, CG)* pairs *(4, 0.9)*, *(6, 0.8)*, and *(8, 0.7)* and expression (11) (or indeed Table 2) has been used to find the corresponding Relative Productivity. Here the *n=4* alternative is the most productive case – the one with the fewest elements.

Table 3. A planning scenario with ***d = 0.833***

n	**CG**	**RLPR**
4	**0.9**	1.13
6	**0.8**	1.08
8	**0.7**	0.99

The above discussion sheds some light on one of the highly debated issues of today: "Is XP a better development method than the traditional approaches?" One of the cornerstones of XP is incremental delivery with a large number of increments. Kent Beck [1] states that the technical premise for this is that the cost of fixing defects rises slowly in time. This is reflected in a slowly increasing Cumulative Gain, in which case productivity is expected to rise significantly with an increasing number of increments as deduced from Table 2. However, the traditional view has been that the cost of fixing defects rises sharply with time [3, 5]. If this is the case then increasing the number of increments could possibly reduce productivity as demonstrated with the planning scenario of Table 3.

It is also interesting to investigate the effects that the exponent *d* exerts on the Relative Productivity. Table 4 exemplifies the effect of varying *d* in a case where the number of increments is fixed at 24. (The values of *d* chosen here correspond to the scale factors in the Basic COCOMO model as seen in Table 1 where the *0.833* value corresponds to the embedded model, *0.893* to the semidetached model, and *0.952* to the organic model which can be viewed as a set of representative values for projects).

Table 4. Relative Productivity ***RLPR*** for *n* fixed at ***n = 24***

	CG Cumulative Gain			
d	**1.0**	**0.9**	**0.8**	**0.7**
0.833	1.70	1.53	1.36	1.19
0.893	1.41	1.26	1.12	*0.98*
0.952	1.16	1.05	*0.93*	*0.82*

An interesting pattern emerges here: The lowest values of *d* yield the highest gain in productivity *(RLPR)*. It is to be noted that a low value of *d* signifies a difficult development context with low overall productivity.

The line of Table 4 for $d = 0.952$ shows that $RLPR < 1$ for the *0.8* and *0.7* values of *CG* indicating that there is a loss in productivity in this case as compared to the one-off development. For the higher values of *CG* there is a productivity gain. High values of *d* (e.g. 0.952) in the COCOMO II model is the result of project parameters of high precedence in project type, flexible documentation, well conducted risk resolution, high team cohesion, and high process maturity [5]. In this type of development instance the Cumulative Gain can be expected to be high as well.

6 Conclusion

IT projects continue to provide challenging tradeoffs for project managers. The tendency to pre-determine many of the key variables (including budget, schedule and scope) prior to project launch leaves managers with few tradeoff options. As a partial response to project failures, many organisations have adopted timeboxing notions to ensure successful completion and delivery. The resulting impact is in creating effortboxes which project managers are expected to balance and control. These can be likened to a *project resource envelope* which determines the key parameters of a

project. To date there has been little effort to understand this phenomenon and to reason about the methods required to control it.

Advice regarding projects is sometimes delivered in the form of the overall effort-box required to increase the likelihood of successful completion (i.e. six people for six months or four people for four months). Table 1 reworks typical estimation models to reflect size in terms of the required effort so that $Size = c\ PM^{d}$. The required effort can equate to the *project resource envelope*, or the effort-box.

The key implication is that the overall project effort is fixed. In other words, resources are determined for the duration of the project. This represents a shift from organisational and project structures that allow the project managers to call on a variety of skills as and when needed thus implying that whilst specialist resources are perhaps no longer obtainable, the project manager is at least dealing with unchanging resource levels. Fixed resource levels eliminate many tasks associated with resource management. Besides reducing the need for comprehensive resource estimation, calculation, levelling, optimisation and smoothing, which often prove to be time-consuming activities, the new arrangement removes the reliance on external availability of resources suggesting that the focus is on the internal project team. The remaining flexibility can be derived from dividing the given total fixed effort into segments for incremental delivery as described in Expression (3) and in trying to maximise the output values made possible through this division.

The paper investigates the implications of working on projects with a fixed-effort level where increments may be utilised to obtain additional, or perhaps the only leverage, for project managers. The productivity gain from incremental development in fixed effort situations comes when *relative productivity RLPR>1* as this is where the productivity is increased compared with a single delivery project.

The most difficult and complex environments with high rates of change and associated low productivity rates provide the best candidates for incremental development. The lowest values of *d* yield the highest gain in productivity *(RLPR)*. High level environments are less likely to show a direct benefit in terms of productivity from adopting an incremental approach (possibly even indicating a loss of productivity compared to the one-off model).

The analytic framework explored in this work provides a new method of reasoning in the increasingly common situations where effort is fixed (along with time, cost and expected scope). The framework enables project managers to explore the impact of adopting incremental delivery practice thereby enabling them to obtain insights and as a result derive leverage from using increments as part of the limited tradeoffs possible under such fixed constraints.

Most of this work makes the simplifying assumption that a uniform development context is maintained throughout the duration of the project. Indeed, significant changes to the context or to the development environment will invalidate most estimation and planning methodologies.

The work reported in this paper can be extended in a number of directions to explore a range of potential avenues. In particular, the authors are interested in addressing a number of emerging challenges that include:

- Collecting empirical data on incremental development to validate the postulated formulation
- Analysing the relationship between incremental delivery and team size
- Relating the findings to factors expressed in alternative estimation models
- Utilising a richer estimation model to conduct deeper analysis of the cost/duration implications of incremental development
- Investigating the relationship between incremental development and the 'evolvability' of systems
- Providing a basis for reasoning about continuation and cancellation of projects (or deliverables) and conducting trade-offs between time, effort and increments in dynamic projects

It is therefore hoped that the work put forward in this paper will serve as an opening for further discussion and investigation. The opening of a new perspective can also stimulate additional insights needed to bridge the gap in the analysis of the economic impact of non-conventional development approaches on the product, process and project.

References

1. Beck, K., 1999. Extreme Programming Explained: Embrace Change. Addison-Wesley.
2. Benediktsson, O. and Dalcher, D. 2003 Effort Estimation in Incremental Software Development. IEE Proc. Softw., Vol. 150, no. 6, December 2003, pp. 351-357.
3. Boehm, B.W.1981. Software Engineering Economics. Englewood Cliffs: Prentice Hall.
4. Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., and Steece, B. 2000. Software Cost Estimation with COCOMO II. Prentice-Hall.
5. Boehm, B. W. and Turner, R. 2003, Balancing Agility and Discipline. Addison Wesley.
6. Clark, B., S. Devnani-Chulani, and B.W. Boehm. 1998. Calibrating the COCOMO II Post-Architecture Model. in Proceedings of ICSE 1998. IEEE Press.
7. Cusumano, M.A. and Selby, R.W. 1995. Microsoft Secrets: how the world's most powerful company creates technology, shapes markets, and manages people, Free Press, New York.
8. Dalcher D. 1994. Falling down is part of Growing up; the Study of Failure and the Software Engineering Community, Proceedings of 7th SEI Education in Software Engineering Conference, New York: Springer-verlag, pp. 489-496.
9. Dalcher, D. 2002. Life Cycle design and Management in Stevens M. (Ed.): Project Management Pathways: A practitioner's Guide, High Wycombe: APM Press.
10. Jones, C. 1994. Assessment and Control of Software Risks. Englewood Cliffs, New Jersey: Prentice-Hall.
11. Jones, C. 2000. Software assessments, Benchmarks and Best Practices. Upper Saddle River, New Jersey: Addison-Wesley.
12. Luqi and. Goguen, J.A. 1997. Formal Methods: Promises and Problems. IEEE Software. Vol 14(1), pp. 73-85.
13. Kan, S.H. 2003. Metrics and Models in Software Quality Engineering, 2nd ed, Pearson Education.
14. Pressman, R.S. and D. Ince, Software Engineering: A Practitioner's Approach. 5 ed. 2000, Maidenhead: McGraw-Hill.

15. Royce, W. 1998. Software Project Management: A Unified Framework. Addison Wesley.
16. Standish_Group. 1995. Chaos 1995. Standish: Dennis, Mass.
17. Standish_Group. 1997. Chaos 1997. Standish: Dennis, Mass.
18. Standish_Group. 2000. Chaos 2000. Standish: Dennis, Mass.
19. Stapleton, J. 1997. DSDM Dynamic Systems Development Method. Addison-Wesley.
20. Thomas, M. 2001. IT projects Sink or Swim, British Computer Society Review.
21. Vliet, H.V. 2000. Software Engineering: Principles and Practice. 2 ed., Chichester: Wiley.

A Comparison of Size Estimation Techniques Applied Early in the Life Cycle

Onur Demirörs and Çiğdem Gencel

Middle East Technical University, Informatics Institute, Ankara, Turkey
{demirors,cgencel}@ii.metu.edu.tr

Abstract. Timing is one of the most critical factors of software size estimation. We need to know quite a bit about the software project to make a meaningful size estimate. However, most of the software estimates should be performed at the beginning of the life cycle, when we do not yet know the problem we are going to solve. In the literature, there are few early size estimation methods. This study demonstrates the results of size estimation methods utilized on a large software intensive military application within the boundary of a project for Request for Proposal (RFP) preparation.

1 Introduction

Software size estimation is an important practical problem in software engineering. Estimation errors are essential cause of poor management which usually results in runaway projects that spiral out of control [1]. Whatever they produce is frequently behind schedule and over budget, and most often they fail to produce any product at all.

There are various approaches to software size estimation. However, size estimation is still considered as “poor” by many people. Today’s metrics and methods are criticized by the general difficulty of the estimation process and the immaturity of the measurement science for the software engineering [2], [3], [4]. Major difficulties can be listed as; unclear theoretical basis, lack of standards on methods, insufficient validation of metrics, inconsistent development phase and product coverage of the estimation models, and estimation timing.

Among those difficulties, estimation timing is one of the most significant. In order to make a meaningful estimate, we need to know quite a bit about the project. However, to be able to respond to contracts and plan in advance, the software estimates should be performed at the beginning of the life cycle, when we do not yet know the sufficient details of the problem. This is the size estimation paradox [5]: Size measurement would be necessary when we do not have enough information and early estimation methods to obtain it. When we can measure with the greatest accuracy, we do not need that information for effort and duration prediction purposes any more.

In fact, most of the recent researches have concentrated on relatively later phases of software development such as software requirements specification, or preliminary design phases by which time; anywhere between 15-40% of the total work effort might have already been expended [5].

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 184–194, 2004.

In this paper, we will discuss the results obtained by the application of different size estimation methods at different phases of a large software intensive military application within the boundary of a project for Request for Proposal (RFP) preparation. The estimations are made during the early phases of the system analysis process. The methods used are Mark II FP, Jones Very Early Size Predictor, and Early Function Point Analysis (EFPA). Among those, Jones Very Early Size Predictor is used to estimate the size of the whole development project at the feasibility study phase. Mark II is used to estimate the size of the whole project after the detailed system level functional requirements are defined, and EFPA is used to estimate a module of the project at consecutive five stages of the requirements analysis phase.

In the second section, the background work on software size metrics and size estimation methods that are applicable to early size estimation are summarized. In the third section, the description of the case project and the application of Mark II FP, Jones Very Early Size Predictor, and Early Function Point Analysis (EFPA) methods to this case are discussed. The results of this study are given in the fourth section.

2 Background

Among the size estimation methods, Function Points Analysis (FPA) [6] is a popular one designed especially for MIS domain software. FPA estimates the size of software in terms of functionality from the users' viewpoint. When compared to technical measures of software such as Lines of Code (LOC), functional size measures can be determined earlier, when details of functional user requirements are identifiable. On the other hand LOC cannot be determined precisely until software coding has been done, a point much later in the development life cycle.

After the original FPA method, variants of the method have been developed to solve the issues of original FPA [7]. These can be listed as IFPUG Function Point Analysis [8], Mark II Function Points [9], Feature Points [10], 3-D Function Points [11], Full Function Points [12] and COSMIC Full Function Points [13]. However, since these methods are designed to estimate size after the software requirements specification is complete, size estimation accuracy decreases if made during the earlier phases such as system analysis and contract preparation.

Another widely referenced metric is Object Points (OP). OP Method was devised at the Leonard N. Stern School of Business, New York University [14]. The concepts underlying this method are very similar to that of FPA, except that objects, instead of functions, are being counted. The software objects may be a Rule Set, a 3GL Module, a Screen Definition, or a Report. Object Points have attracted interest as Object Oriented Analysis and Design methodologies became more popular. Later a well known cost estimation model, COCOMO II, has recommended Object Points as a way of getting an early estimate of the development effort for business oriented information systems [2]. Moreover, it can be easily understood by the estimators and the automation of this method is possible. There exists many variants of Object Points and there is no standard established for counting. In addition, its usage is restricted to object-based ICASE environments [15].

In the literature, there exist a few methods which have been developed especially for size estimation prior to software requirements phase is completed. One group of these methods (also called as "Rules of Thumb") makes estimation based on experi-

ence or on a speculative basis. One of these methods creates a very rough approximation of FP totals long before requirements are complete [16]. Project characteristics and complexity were considered by including software development environment factors that adjusted a preliminary size-based estimate. However, these methods are stated to be very inaccurate for serious cost-estimating purposes.

In this context, another early estimation method; Early Function Point Analysis (EFPA) technique was developed in 1997 [17] and subsequently refined [18],[19] to estimate the functional size to be developed or maintained in the early stages of the software project life cycle. The designers of this method stated that "EFPA is not a measurement alternative to FPA method, but only a fast and early estimate of them, obtained by applying a different body of rules". EFPA combines different estimating approaches in order to provide better estimates of a software system size. This method makes use of both analogical and analytical classification of functionalities. In addition, it lets the estimator identify software objects at different levels of detail [20]. Since IFPUG FPA method is applicable to MIS software, so is EFPA.

After that, in 2000, a new size estimation technique, Early & Quick COSMIC-FFP [5] was designed by the same research group. This early method is based on the present COSMIC-FFP design [21] to help estimate functional size of a wide range of software (i.e. real-time, technical, system and MIS software) at early stages of the development life cycle.

3 Case Study

3.1 Description of the Case

We applied the estimation methods to a project which targeted the requirements elicitation for a model Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance - C4ISR sub-system for the Turkish Land Forces Command. The project outcomes formed the major parts of the Request for Proposal (RFP) currently issued by the Turkish Armed Forces. Some characteristics of the related project are given below in order to better describe the application of size estimation methods.

In this project we applied the requirements elicitation approach that we defined in an earlier study [22]. The approach emphasizes business process modeling to elicitate requirements. The life cycle we utilized is depicted in Fig. 1.

While modeling the business processes; organizational charts, function trees, and Extended Event Driven-Process Chain (eEPC) diagrams were used as basic process modeling notations. Each lowest level sub-process was modeled in terms of its processes, process flow, inputs, outputs, and the responsible bodies. Totally, 295 distinct diagrams consisting of 1270 functions were created to model existing business processes of different levels of organization units by using the eEPC notation.

The project was started in October 2002 and completed in thirteen months. The project staff consisted of 11 part-time persons. The total effort spent during the project was 24 person-months. The project outcomes formed the major parts of the Request for Proposal currently issued by the Turkish Armed Forces.

By using the business process models generated during this project, in this study, we used Mark II FP, Jones Very Early Size Predictor, and EFPA methods to estimate

size of the project. Among those, Jones Very Early Size Predictor is used to estimate the size of the whole development project at the feasibility study phase. Mark II is used to estimate the size of the whole project after the detailed system-level functional requirements are identified. Lastly, EFPA is used to estimate a module of that project at five consecutive stages of the requirements analysis phase starting after the feasibility study until the system level requirements are generated (see Fig. 1).

Fig. 1. Requirements analysis life cycle

Mark II FP estimation was performed by the RFP preparation project team. The staff involved in the Mark II FP size estimation process consisted of 4 software analysts who have the domain knowledge. It should be noted that, although they have experience in using the method, they are not experts. One of the project team software analysts, who is also an author of this paper performed EFPA and Jones Very Early Size Predictor estimations.

3.2 Application of the Methods

The applications of the methods are described below:

Mark II Function Points. Mark II FP Method aims to measure the information processing. This method views the system as a set of logical transactions and calculates the functional size of software based on these transactions [9], [23].

In this study, the software requirements are generated from business process models with respect to 11 different subsystems as shown in Table 1. Original names of the subsystems and modules in the project are not given due to security constraints.

Table 1. Size estimates of subsystems by Mark II

Subsystem	Module	Unadjusted FP
A	A1	2886.64
	A2	4882.10
	A3	9281.55
B		8.48
C		185.34
D		3344.96
E		878.31
F		386.66
G		1000.00
H		1000.00
I		1000.00
J		200.00
K		400.00
Total Project Size		25454.04

While making the estimation by Mark II FP, the size of each subsystem is estimated and then summed to compute the size of the whole development project. The size of the software to be contracted for the whole development project is estimated as 25,454 FP by the project team. The difficulties faced during Mark II FP estimation are discussed in the following paragraphs.

Since, the logical transactions can be correctly identified only after the completion of the software requirements specification phase, and we are in an earlier stage where each requirement might involve of more than one transaction; we classified the requirements into three categories according to the kind of transactions it may involve. These categories are “Copying”, “Preparation”, and “Determination”. “Copying” involves the following transactions: viewing input(s), inserting these input(s) into the database, CRUD operations (Create, Read, Update, Delete) on these data in the database, and viewing the output(s). “Preparing” differs from “Copying” in that the user(s) may add other input data by means of input form(s). For the requirements which end up with the verb “Determine”, more transactions are involved in addition to “Preparing” category. In fact, for most of the requirements, the type of these transactions could not be determined definitely due to their high abstraction levels.

For each logical transaction, Input Data Element Types, Data Entity Types Referenced and Output Data Element Types are determined. However, for some inputs, outputs, and processed entities, we have insufficient information about their Data Entity Types (DETs). Therefore, we made some assumptions about the number of

DETs of some transactions. The percentage of such transactions is about 60% of the overall.

The software requirements are generated from the business process models with respect to 11 different subsystem types. However; for 5 of the subsystems, i.e. Subsystem G, H, I, J, K (see Table 1), we could not make size estimation using Mark II FP methodology, since the requirements could only be determined at a very high abstraction level. Thus, we had to use expert opinion to estimate their sizes. Most of these requirements fall into the "Determining" category that we have just described. However, for these requirements, not only the DETs, but also the type of transactions could not be determined. The percentage of the number of such requirements to overall is 2.2%. The percentage size of the subsystems involving these requirements to the whole project is found to be 14.1%.

The effort needed to make Mark II FP estimation for the whole project is found to be 131 person-hours.

Jones Very Early Size Predictor. This is an estimation method developed by Capers Jones to be used for very early size approximation.

The method utilizes taxonomy for defining software projects in terms of "Scope", "Class", and "Type" in order to identify a project when entering information into the software cost estimation tools (see Table 2). The taxonomy is then used for predicting the size of the software by means of the following formula:

$$\text{Size} = (\text{Scope} + \text{Class} + \text{Type})^{2.35} \tag{1}$$

In this study, by choosing the scope as "compound system", class as "military contract", and type as "process control", the size of the whole development project is predicted as 4542.67 FP.

Table 2. Taxonomy for defining software projects

Scope:	**Class:**	**Type:**
1) all that needs to be written is a function.	1) individual software	1) nonprocedural
2) module	2) shareware	2) web applet
3) reusable module	3) academic software	3) batch (not database)
4) disposable prototype	4) single location – internal	4) interactive
5) evolutionary prototype	5) multi location – internal	5) interactive GUI
6) standalone program	6) contract project – civilian	6) batch database
7) component of a system	7) time sharing system	7) interactive database
8) release of system	8) military services	8) client/server
9) new system	9) internet	9) mathematical
10) compound system	10) leased software	10) systems
	11) bundled software	11) communications
	12) marketed commercially	12) process control
	13) outsourced contract	13) trusted system
	14) government contract	14) embedded
	15) military contract	15) image processing
		16) multimedia
		17) robotics
		18) artificial intelligence
		19) neural net
		20) hybrid: mixed

Early Function Point Analysis (EFPA). Early FP technique [17] uses both analogical and analytical classification of functionalities. This provides estimating a software system size better. The estimator may have knowledge at various levels of detail about different branches of the application; from almost nothing to very detailed. In EFPA, the estimator can identify software objects at different detail levels, which makes it possible to make use of all the information the estimator has on a particular application [20]. The software objects in EFPA are [17]:

– Functional Primitives,
– Macrofunctions,
– Functions,
– Microfunctions,
– Logical Data Groups.

Functional Primitives: The elementary processes of the standard FP Analysis (External Input, External Output, External Inquiry).

Macrofunctions (MF), Functions (F), and Microfunctios (mF): Different aggregation of more than one Functional Primitive (fP) at different detail level.

Logical Data Groups (LD): Standard Logical Files with levels suitable for aggregation of more than one logical file. There is no differentiation between "external" and "internal" data.

In EFPA, each "object" is assigned a set of FP values as shown in Table 3.

Table 3. Elements of the EFPA method and their associated values (Unadjusted FP) [19]

LD	Min	Avg	Max	mF	Min	Avg	Max
Simple	5	6	7	mF	16	18	20
Ave.	8	9	10	**F**	**Min**	**Avg.**	**Max.**
Complex	13	14	15	Small	45	56	67
Low Multip.	14	18	22	Med.	73	91	109
High Multip.	27	39	51	Large	106	133	160
fP	**Min**	**Avg**	**Max**	**MF**	**Min**	**Avg**	**Max**
PI	4	5	7	Small	151	215	280
PO	5	6	8	Med.	302	431	560
PQ	4	5	7	Large	603	861	1119

EFPA entails a certain degree of subjectivity due to the fact that "its reliability is directly proportional to the estimator's ability to recognize the components of the system as part of one of the proposed classes" [19]. Thus, the developers of this method suggested that the expression of user requirements should be formalized as much as possible [19]. Therefore, at the beginning of this study, we believed that business process models may help this formalization.

In this study, we selected one of the modules of a subsystem of the whole development project module (Subsystem A-Module A1) and estimated the size of this Module by applying EFPA.

Since the business process models becomes more detailed as the requirements elicitation process proceeds, we applied EFPA to five different stages of the require-

ments analysis process, starting after the feasibility study until the system level requirements are generated. Thus, the estimates provided by this method are denoted as "Stage-0", "Stage-1", "Stage-2", "Stage-3", and "Stage-4" depending on when the estimation is made during the requirements analysis phase (see Fig. 1).

The size estimates for each stage by EFPA are summarized in Table 4.

Table 4. EFPA size estimates for consecutive stages

	Unadjusted EFP's		
Stage	**Min.**	**Avg.**	**Max.**
Stage 0	658	940	1222
Stage 1	780	1048	1318
Stage 2	1204	1461	1796
Stage 3	1454	1793	2155
Stage 4	1707	2089	2554

4 Comparison of the Methods

Various benchmarking models, which take into account a set of quality criteria, exist for comparing the size estimation methods [20]. Depending on the needs of the organization as well as the circumstances, an estimation method can be evaluated as optimal or not so good. In this study, our aim is not to select one of the methods as being better than others, but rather just to evaluate those methods' applicability for early size estimation in a case study.

Mark II is used to estimate the size of the whole project after the detailed system-level functional requirements are identified. The size of the software to be contracted for the whole development project is estimated as 25,454 FP. The effort needed to make Mark II FP estimation for the whole project is found to be 131 person-hours.

Jones Very Early Size Predictor is used to estimate the size of the whole development project at the feasibility study phase. The size of the whole development project is predicted as 4542.67 FP. Although the time needed to make this estimation is in the order of minutes, the estimate is found to be very rough with respect to Mark II FP estimate.

Lastly, EFPA is used to estimate a module of that project at five consecutive stages of the requirements analysis phase starting after the feasibility study until the system level requirements are generated (see Table 4). By Mark II FP method, the number of transactions is found to be 154 and the size is estimated as 2886.64 Unadjusted FP's for the same module. The time of Mark II estimation corresponds to Stage 4 of EFPA.

The results of EFPA and Mark II estimations are not directly comparable with each other since different metrics were used. EFPA uses the same metric as IFPUG FP's. Therefore, in order to compare EFPA and Mark II estimates, a conversion of Mark II FP size estimate to IFPUG FP size estimate was performed.

Symons (1999) defined the average size relationship between Mark II FP and IFPUG FP. For the projects, size of which are above 1500 IFPUG Unadjusted FP's or 2500 Mark II FP's, the ratio can be found by the following formula [24]:

$$\frac{MarkII\_FP}{IFPUG\_FP} = 0.16 \times \frac{\text{No of Entity References}}{\text{No of Entity Types}} \tag{2}$$

For the whole system; the average number of references of each entity type is found to be 9. Thus, by using the above formula, the ratio of Mark II FP size to IFPUG FP size is calculated as 1.44. Accordingly, the size of Module A1 is found as 2004.61 IFPUG FP's.

The size estimates by EFPA are compared with the estimates by Mark II method (converted to IFPUG FP's), and the relative percentage errors are calculated. The results are given in Table 5.

As seen in this table, as we proceed, the relative error of the unadjusted EFP with respect to Mark II FP decreases. If we compare EFP and Mark II FP at Stage 4, during which the same system level requirements and business process models are used for both methods, the relative error was found to be between -14.85 % and +27.41 %.

Table 5. Size estimates by EFPA and the relative errors with respect to Mark II FP estimate at consecutive stages

	Relative Error (%)		
Stage	**Min.**	**Avg.**	**Max.**
Stage 0	-67.18	-53.11	-39.04
Stage 1	-61.09	-47.72	-34.25
Stage 2	-39.94	-27.12	-10.41
Stage 3	-27.47	-10.56	7.50
Stage 4	-14.85	4.21	27.41

One of the results of EFPA estimation showed that, while applying the method, business process models are very useful to identify software objects at different detail levels. Five stages involve five different groups of business process models from which the software objects are identified and according to which the size of Module A1 is estimated.

Another result is that, for Module A1; the effort needed to make EFPA is found to be 24 person-hours, whereas it took 35 person-hours to make Mark II estimation.

In addition, we observed that a structured estimation process should be defined and a standard guideline, such as a measurement manual, must be produced. This will ensure that for all projects, consistent and reliable size estimates can be made by different users.

5 Conclusion

According to the results of this study, it can be concluded that the size estimation by Jones Very Early Size Predictor is very rough. If we have used the highest assignments for Scope, Class, and Type, the maximum size a project can have would be approximately 7675 FP. This means that this method can not be used especially for large projects.

Mark II is used to estimate the size of the whole project after the detailed system level functional requirements are defined. While making estimation, some difficulties were faced as this method is designed to estimate size after the software requirements specification is complete. The abstraction levels of system level functional requirements differ. Therefore, some assumptions on the kind of transactions and the number of DETs should be made while making Mark II FP estimation. The accuracy of the method decreases as the abstraction level of the requirements gets higher. In addition, for some requirements, the method could not be used at all. Thus, if used earlier in the life cycle, Mark II FP method can be used by making some assumptions. However, this may result in under or over estimation of the project size.

Lastly, EFPA, which is designed especially for early size estimation, is used to estimate a module of the project at five consecutive stages of the requirements analysis phase starting after the feasibility study until the system level requirements are generated. The results showed that at the earlier stages, the relative error of this method increases from 4.21% to -53.11% on the average. In their study [19], the designers of EFPA presented data gathered by a number of EFP forecasts for projects in which the actual values were then made available. In that study, the project sizes vary between 154 FP to 1434 FP and the tendency by which the average deviation between the forecast and the actual value is found to be below 10%. The greater deviations in our study may be due to inappropriateness of the assignments shown in Table 3 for large projects. We suggest that these assignments can be subject to improvement on the basis of gathering more data on larger projects.

In addition, since the reliability of the EFPA is directly proportional to the estimator's ability to "recognize" the components of the system as part of one of the proposed classes, EFPA method entails a large degree of subjectivity. Therefore, the developers of this method suggested that the expression of the user requirements should be formalized as much as possible in order to simplify and optimize the forecast of the project's size [19]. This study showed that business process models help this formalization and simplify identifying software objects at different detail levels. We observed that the automation of this method is possible by associating this method with the business process model components.

Another result is that the effort needed to make EFPA is found to be about 31% less than the effort to make Mark II estimation for the same module.

All of these metrics and methods can be used for early size estimation. However, they all have their restrictions. Jones Very Early Size Predictor is far too inaccurate for serious estimation purposes, but it had the virtue of being usable for small projects before any other known form of sizing is possible. Mark II FP, which is designed to estimate the size after the software requirements specification is complete, can be used with some assumptions and with expert opinion methods' support in earlier phases. EFPA can give better results if the assignments of the method are improved on the basis of gathering more data on other projects and definition of the requirements is formalized in order to decrease its subjectivity.

It is for sure that early size estimation is an area demanding further research. New methods, metrics and guidelines are required to make accurate size estimations early in the life cycle as well as studies to validate the suggested metrics and models.

References

1. Glass, R.L.: Facts and Fallacies of Software Engineering, Addison Wesley (2002)
2. Hughes, B.: Practical Software Measurement, McGraw-Hill (2000)
3. Fenton, N.: Software Measurement: A Necessary Scientific Basis, IEEE Transactions on Software Engineering, Vol.20, No.3, March (1994)
4. Fenton, N.E. and Pfleeger, S.L.: Software Metrics: A Rigorous and Practical Approach, Second Edition, International Thomson Computer Press (1996)
5. Meli, R., Abran, A., Ho, V.T., Oligny, S.: On the Applicability of COSMIC-FFP for Measuring Software Throughout Its Life Cycle, Escom-Scope (2000)
6. Albrecht, A.J. and Gaffney J. E.: Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, IEEE Transactions on Software Engineering, vol. SE-9, no. 6, November (1983)
7. Symons, C.: Come Back Function Point Analysis (Modernized) – All is Forgiven!), Proc. of the 4th European Conference on Software Measurement and ICT Control, FESMA-DASMA 2001, Germany (2001) 413-426
8. IFPUG: Counting Practices Manual rel. 4.1, International Function Point Users Group, Westerville, OH (1999)
9. Symons, C. R.: Function Point Analysis: Difficulties and Improvements, IEEE Transactions on Software Engineering, vol. 14, no. 1, January (1988)
10. Jones, T.C.: A Short History of Function Points and Feature Points, Software Productivity Research Inc., USA (1987)
11. Whitmire, S.A.: 3D Function Points: Scientific and Real-time Extensions to Function Points, Pacific Northwest Software Quality Conference (1992)
12. Abran, A., St-Pierre, D., Maya, M., Desharnais, J.M.: Full Function Points for Embedded and Real-Time Software, UKSMA Fall Conference, London (UK), October 30-31 (1998)
13. Abran, A.: COSMIC FFP 2.0: An Implementation of COSMIC Functional Size Measurement Concepts, FESMA'99, Amsterdam, October 7 (1999)
14. Kauffman, R. and Kumar, R.: Investigating Object-Based Metrics for Representing Software Output Size, Conference on Information Systems and Technology (CIST), in the INFORMS 1997 Annual Conference, San Diego May (1997)
15. Sırakaya, H.S.: A Comparison of Object Oriented Size Evaluation Techniques, A MSc Thesis, Informatics Institute of the Middle East Technical University, Ankara, Turkey January (2003)
16. Jones, T.C.: Estimating Software Costs, McGraw-Hill (1998)
17. Meli, R.: Early and Extended Function Point: A New Method for Function Points Estimation, IFPUG-Fall Conference, September 15-19, Scottsdale, Arizona, USA (1997)
18. Meli, R.: Early Function Points: a new estimation method for software projects, ESCOM 97, Berlin (1997)
19. Santillo, L. and Meli, R.: Early Function Points: some practical experiences of use, ESCOM-ENCRESS 98, Roma, Italy, May 18 (1998).
20. Meli, R. and Santillo L.: Function Point Estimation Methods: A Comparative overview, FESMA 98 - The European Software Measurement Conference - Amsterdam, October 6-8 (1999)
21. The Common Software Measurement International Consortium (COSMIC): FFP, version 2.2, Measurement Manual (2003)
22. Demirörs, O., Gencel, Ç. Tarhan, A.: Utilizing Business Process Models for Requirements Elicitation, 29th Euromicro Conference, 1-6 September, Antalya, Turkey (2003)
23. United Kingdom Software Metrics Association (UKSMA): MK II Function Point Analysis Counting Practices Manual Version 1.3.1 (1998)
24. Symons, C.: Conversion between IFPUG 4.0 and MkII Function Points, Software Measurement Services Ltd., Version 3.0 (1999)

Model and Process for Timely Progress Reporting in Use Case Driven Software Development

Sungwook Moon[1] and Joa Sang Lim[2]

[1] ComponentBasis Co., Ltd.
137-070, 7F Kyungmok Bldg., 1354-5, Seocho-dong, Seocho-gu, Seoul, Korea
swmoon@componentbasis.com

[2] Division of Media Technology, Sangmyung University
110-743, 7 Hongji-dong, Jongro-gu, Seoul, Korea
jslim@smu.ac.kr

Abstract. Iterative development is often the practice that most object oriented and component based projects adopt. It entails, however, dramatically increased managerial efforts due to seemingly never-ending user requirements that certainly add to complexity of project management. We emphasized the importance of progress, operationalized as a function of size and fault. Moreover, it is important that the progress be reported regularly and timely which necessitates collection of relevant data and more elaborated managerial process. We proposed progress estimation model and data model to automate data collection and efficient process for planning and change management on which project support systems were built and reported in this study. In the application of this newly designed management process and system to enterprise-wide projects, project risks appeared to be mitigated to a considerable extent with timely reporting of project progress. As a result, the resource planning and scheduling to control the change requirements was made available with ease. These practical managerial processes could make it possible to identify and resolve critical issues and risks at an early stage and eventually lead to high quality product.

1 Introduction

Despite remarkable advances in the theories and techniques of software engineering over the past decades, it is still mythical to witness that software projects undergo cost and schedule overruns and often come to an end with user requirements unfulfilled. The Standish Group [1] reported that only 16.2% of the surveyed projects completed on-time and on-budget while 52.7%, completed unsuccessfully and 31.1%, even cancelled. Average time overruns were 222% of the original time estimates and only 61% of originally specified features and functions were implemented on the delivered projects. Conceived ambitiously to tackle this myth and thus, more often favored in the object-oriented and component based development, the iterative approach seems to add to rather considerable difficulty by unfreezing user requirements, which requires more elaborated planning and resource utilization. Then do we have a key performance indicator that is simple but comprehensive and robust enough to

T. Dingsøyr (Ed.): EuroSPI 2004, LNCS 3281, pp. 195–206, 2004.

alarm relevant risks and guide project managers to take any preventive actions? This paper takes a quantitative approach to the progress that can be used to control schedule and quality of iterative software development. FPA (Function Point Analysis) [2] was employed to estimate the size and the progress of individual use cases for the software project. The completed work (i.e., use case) is subject to careful inspection and the progress is adjusted accordingly. Such operationalization of the progress as a function of size and defect would certainly alleviate the misconception that the schedule can be opted for the quality. Timely reporting of the progress would be more helpful and this requires well-coordinated functioning managerial processes of job scheduling, progress management, change management and configuration management among others. With the progress model and its related development processes, we proposed a project supporting systems where project managers could be equipped with more timely information and what-if capabilities on progress to control software projects as suggested by decision support proponents [8].

2 Iterative Measurement of Progress

2.1 Nomenclature

P	progress of software development (use case based)
F_D	functionality delivered in FP (Function Points)
C	degree of completion
F_T	total functionality in FP
F_i	FP of the *i*th use case
r^k	effort ratio for the development phase k
t	staff time required
t'	staff time spent
t_i	total staff time required to develop the *i*th use case
t_i^0	staff time initially scheduled for the *i*th use case development
θ_{ij}	staff time required to fix the *j*th fault of the *i*th use case
t_{FP}	staff time required to complete one FP
c_{ij}	coefficient for degree of completion for the *i*th use case and the *j*th fault fix
C_i^k	degree of completion of the *i*th use case at the development phase k

2.2 Progress

The use case based iterative approaches to system development have been quite a common practice. Unlike the traditional waterfall method, however, the approach assumes that user requirements are subject to change and evolutionary toward the later stages of software development. Therefore a somewhat different viewpoint is required to measure it. Equation (1) defines the progress as the extent to which the system functionality is implemented and delivered up to a certain point of time. The

evolutionary user requirements are taken into consideration as in equation (1) by multiplying the delivered functionality by the degree of completion over iterations

$$P = \frac{F_D C}{F_T} \ . \tag{1}$$

Although there exist a number of methods for size and effort estimation (e.g., LOC), we opted for FPA (Function Point Analysis) [2], [12] due to their early availability over the life cycle of system development. It should be noted that the unit of progress is individual use case and FPA needs some elaboration to be employed in the use case driven development. Firstly, size and efforts are carefully estimated for the use cases with the entities that are shared. Secondly, user requirements are subject to variation over iterations that could change the baseline as in equation (1). More importantly, the change in any use case could claim further efforts once delivered in the earlier iteration. This is the rationale for the degree of completion in equation (1).

2.3 Degree of Completion

Despite its importance, controlling software quality has been neglected [11] and use case driven quality assurance has been far lacking. Software inspection is used to control the quality of use case driven software development. Once claimed delivered, any use case goes under review to find any errors and thus to minimize potential defects [3]. From equation (1), we define the degree of completion as follows:

$$C = \frac{t'}{t} \ . \tag{2}$$

Suppose that the staff time initially scheduled for the ith use case is $t_i^0 = t_{FP} F_i$, the total staff time required to complete the ith use case is computed as:

$$t_i = t_i^0 + \sum_{j=1}^{n} \theta_{ij} = t_{FP} F_i + \sum_{j=1}^{n} \theta_{ij} \ . \tag{3}$$

Here t_{FP} is initially obtained from total staff time and total functionality. In some cases, it is affected by the technological environment of the project. Upon completing each iteration, t_{FP} is recalculated as a function of the progress, that is:

$$t_{FP} = \frac{\Delta t}{F_T \Delta P} \ . \tag{4}$$

From the equations (2) and (3), degree of completion for the ith use case can be expressed as follows:

$$C_i = \frac{t_i^{0'} + \sum_{j=1}^{n} \theta_{ij}'}{t_i^0 + \sum_{j=1}^{n} \theta_{ij}} \tag{5}$$

Here let c be a coefficient of degree of completion, then $t_i^{0'} = c_i\, t_i^0$ and $\theta_{ij}' = c_{ij}\theta_{ij}$. If the jth fault is fixed, $\theta_{ij}' = \theta_{ij}$ and otherwise $\theta_{ij}' = 0$. Finally, the degree of completion is computed using the following equation:

$$C_i = \frac{c_i t_{FP} F_i + \sum_{j=1}^{n} c_{ij}\theta_{ij}}{t_{FP} F_i + \sum_{j=1}^{n} \theta_{ij}} \tag{6}$$

2.4 Integration

Remember that equation (1) defined the progress as the ratio of the functionality delivered to the total functionality. By replacing the degree of completion as earlier computed, the progress for each iteration becomes:

$$P = \frac{\sum_i \left[F_i \left(\sum_k r^k C_i^k \right) \right]}{F_T} . \tag{7}$$

Equations (6) and (7) are taken together and the progress is computed as follows:

$$P = \frac{\sum_i \left[F_i \left(\sum_k r^k \frac{c_i^k r^k t_{FP} F_i + \sum_{j=1}^{n} c_{ij}^k \theta_{ij}^k}{r^k t_{FP} F_i + \sum_{j=1}^{n} \theta_{ij}^k} \right) \right]}{F_T} . \tag{8}$$

3 Development Process

It is also important to know the progress in time rather than simply to estimate it. Timely reporting of progress would require enormous efforts to collect required data and thus, automated data collection and efficient development process are essential [7]. Some of the issues in relation to process and data gathering for the progress model are discussed herein (for details, see [16]).

- Accurate scheduling of staff time
- Individual reporting of work progress
- Change management and fault removal

3.1 Scheduling

Over the last decade, the use case driven development has been well adopted in object orientation [6]. Use cases are useful to partition the whole system into manageable sizes despite some concerns about their granularity. Figure 1 shows that some project

base data (e.g., productivity, effort ratio) are required to be set for further scheduling. Then, we identify use cases as a group of events or equivalently functions, based on which FP is computed [2]. This serves as a baseline for the size estimation of use case and overall software development project. Once the development cycle is performed, the project base data may be adjusted. Suppose that the size of a use case (F_i) amounts to 25 FPs and productivity (t_{FP}) is estimated as 20 man-hours. Then, the efforts required to develop the use case are 500 man-hours (= 25 FPs x 20 man-hours). The total efforts of staff time (i.e., 500 man-hours) may be allocated to development phases in reference to the effort ratio (r^k). For example, we may refer to the weights of 30%:20%:20%:10% respectively for (1) analysis and design, (2) implementation, (3) test and (4) project management. We often reserve the remaining 20% of time just in case to adjust any inaccurate scheduling. It is important to note that some efforts should also be set aside for review to prevent any serious faults from rolling over to the later part of system development. The data collected over those processes would certainly provide useful information as to productivity, the extent of delivery and the current state of use cases.

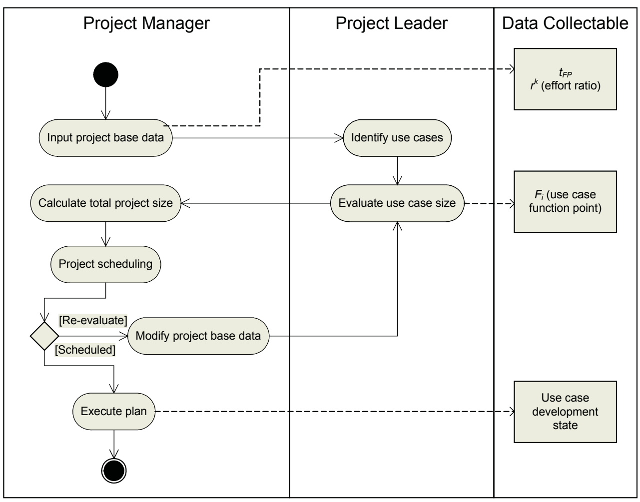

Fig. 1. Scheduling Process and Related Data Collection

3.2 Work Orders and Reporting

Given the size and effort estimation as earlier discussed, workers (e.g., developers) are given an order (e.g., use case development) based upon use case function points. Once the order is fulfilled and the artifacts (e.g., source codes) are submitted by the worker in charge to the quality engineers for review (Figgure 2). Regardless of the job completion, the workers are required to report their daily time spending (t') (see the innermost 'daily reporting' loop of Figure 2). The artifacts are subject to thorough reviews until regarded as 'delivered' or the workers do the job by taking the middle

loop after the daily reporting loop. Over this review process, the coefficient of degree of completion (c_{ij}) and staff time required to fix any errors (θ_{ij}) are generated for equation (8). Any possible change requests over the iteration may take the outermost loop, which adds to the time spent for the use case (for details, see section 3.3). Further issuing work orders would be ceased only in case where there exist no errors in the checked-in artifacts. Then the use case is flagged as 'job done' and the delivered function points are adjusted accordingly.

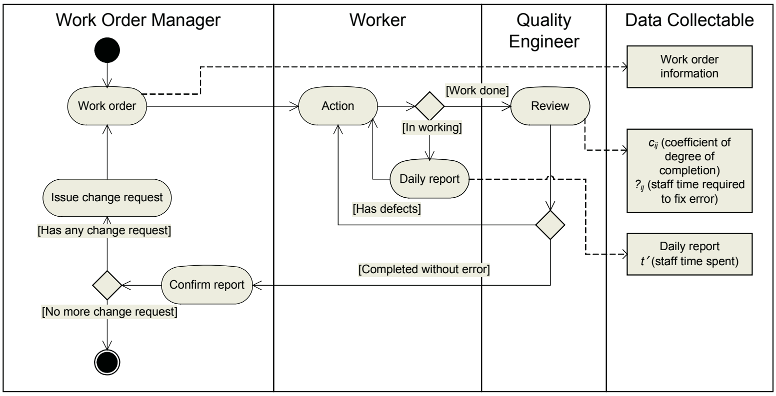

Fig. 2. Work Order Management Process and Related Data Collection

3.3 Change Management

The 'moving target' problem bears well out the nature of user requirements being hardly stable and easily changeable. This urges more elaborated change management especially with the iterative development process. The change requests may arise from three different sources, that is, (1) error faults (F) as found and requested mostly by the reviewers (e.g., UML notation errors), (2) improvement requests (I) often suggested by the users to make the as-is systems better (e.g., changes in discount rates) and (3) brand-new requests (N) that may arise mainly due to addition of new business processes. Any of those requests should ride on the processes for approval of either reviewers (F, I) or the change control board (N) as depicted in Figure 3. Any accepted change requests are entitled to call upon work orders. In this case, FP for the work needs to be computed and considered into the size.

3.4 Progress Reporting

Now we have all required data to analyze and compute the progress as defined in equation (8). We believe that this analysis and reporting should be automated to make the data available to the project mangers on a near real-time basis (e.g., daily). Those progress analyses and related key status information (e.g., fault density, quality & productivity) would be more robust and dependable as accumulated over the iterations, sufficiently enough for the managers to conduct various what-if simulations.

This would also certainly keep the managers most up to dated as to the status of the project and help them to take right preventive actions as early as possible.

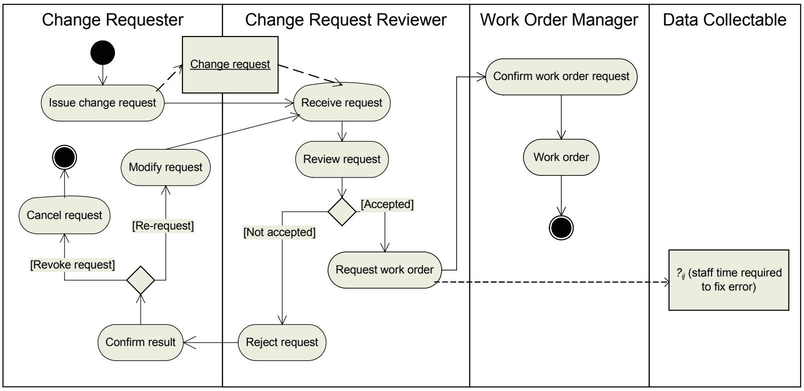

Fig. 3. Change Request Process and Related Data Collection

4 Applications

4.1 Data Modeling

Given the model and the processes as discussed above, a data model is constructed as in Figure 4, which will be used to develop the project support systems. The *Project* entity contains *EffortRatio* (r^k) and total FP (*Project.totalFunctionPoint*, sum of all use case function points). The *Project.manhourPerFunctionPoint* denotes the man-hours per FP (t_{FP}), that is, a productivity indicator as suggested in [4].

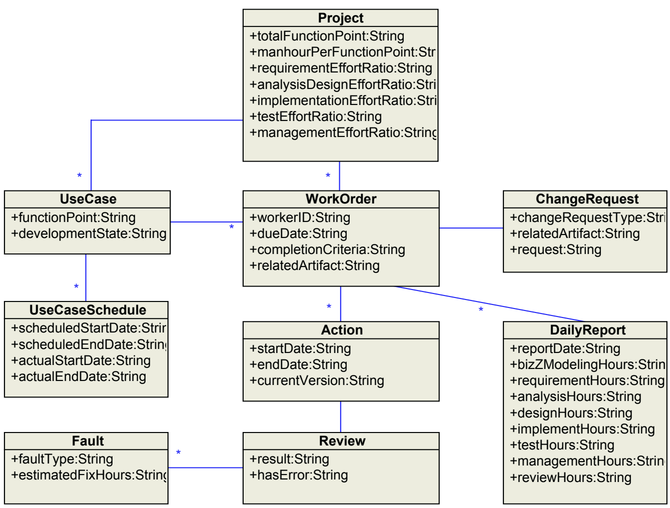

Fig. 4. Key Attributes and Data Model for the Progress Model and the Development Process

The *UseCase* entity keeps data on each use case such as FP and a flag to indicate its state (*UseCase.developmentState*) (e.g., in-progress, under review, delivered). All time logs for use cases are saved in the *UseCaseSchedule* entity and thus, any controlling action can be taken by analyzing the deviation between the scheduled and the actual. The *WorkOrder*, *Action* and *Review* entities keep records of job orders, review activities and their completion related metrics. The review results are written to the *Fault* entity. A work order is given a due date (*WorkOrder.dueDate*) depending on the size of FP (*UseCase.functionPoint*). The work order records all man-hours in the *DailyReport* entity. The *ChangeRequest* entity holds a flag on three different types (F, I, N) (*changeRequestType*).

4.2 Project Support Systems

Although decision aids have been suggested against any likely biases and errors of human decision making (e.g., decision support systems, [8]), project managers appear to be left all alone without much help. Recently there arises some research to address this issue [9], [10]. We developed the Project Support Systems (PSS) where the project managers can be provided with right information to make right decisions [16]. The PSS was designed to run on the web [13] and implemented in Java and JSP (Java Server Page). Data were saved in the MS-Access and the server was a personal computer hooked up to the Internet.

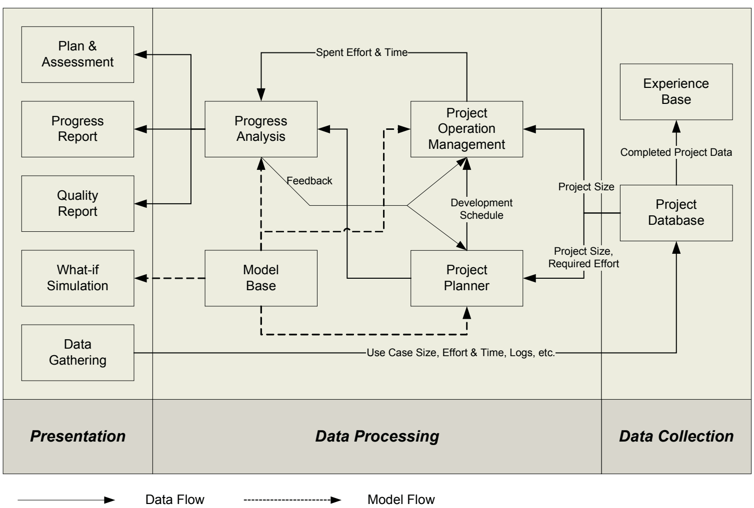

Fig. 5. The High Level Architectural View of Project Support Systems

Figure 5 shows the high-level architectural view of PSS which contains three explicit layers. All relevant data (e.g., FP, use case list) required for progress reporting are collected in the data model of the rightmost *data collection* layer. The second *data processing* layer contains four components: (1) the *project planner* which helps to put together an initial development plan with size and effort information, (2) the *project*

operation management which assists such operations as work order, daily reporting and change requests, (3) the *progress analysis* which relies on the earlier two components to analyze the project progress and lastly (4) the *model base* which contains useful analysis tools and models for the other components. Over the feedback path, scheduling and daily operations can be aided by the *progress analysis* component. The project manager interacts with the last *presentation* layer which provides five functionalities of (1) plan and assessment, (2) progress report, (3) quality report, (4) what-if simulations and (5) data gathering. With the modeling capabilities and data provided by the two lower layers (i.e., *data collection*, *data processing*), the project manager are empowered to plan the goals and detect early the project risks that may threat product quality and schedule. The progress report shows the state of the project and compares the planned and the actual. Project manager may also conduct what-if simulations on scheduling and productivity (for details, see section 4.3). The project manager may consult the quality report which shows how densely the use case contains faults and how fast the faults are fixed. The fault density is compared to the industry level, so the project manager can monitor the health of the project. As the unit of measurement is individual use case, the PSS can web-publish performance reports by workers or summarized reports by teams.

4.3 What-If Simulations

Schedule. Some of the what-if questions that may be raised by the project manager include: (1) what if we step up the productivity by 10%?, (2) what if we slow down the productivity by 10%?, (3) what if we let the system boundary expanded by accepting the change requests?, (4) what if we keep pacing at the current speed of development? and so forth. As the PSS keeps track of staff time required to complete one function point (t_{FP}) and fault density, the impact of productivity related what-if queries can be easily simulated and thus, we can forecast when the project would end. For the additional change requests, the required staff time to fix (θ_{ij}) is used to estimate their impact on scheduling.

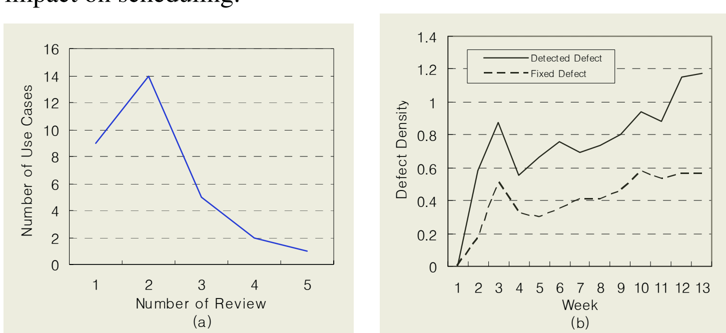

Fig. 6. Sample Quality Reports for a Case Project (31 use cases & 1,500 FPs): (a) use case vs. review rounds over requirement artifacts and (b) weekly distribution of defect density (defined as the staff time required to fix defects per 10 FPs) (The solid and the dotted lines denote detected defects and fixed defects respectively)

Quality. The progress model is two dimensional and thus, any scheduling related what-if questions should be analyzed in consideration of the quality factor (i.e., fault density & fault removal). Figure 2 (a) shows that some use cases needed as many as five rounds of reviews. This indicates that faults should be minimized to reduce their detrimental effect on progress and productivity. Figure 2 (b) presents fault density and removals over weeks. As the size of use cases varies, we normalized fault density per 10 FPs. For example, one fault for a larger use case of 50 FPs is not equal to the equal number of fault for a smaller one of 10 FPs. In this case, the fault density of the bigger use case is one fifth of the smaller one.

5 Discussion

Despite enormous efforts to improve the process (e.g., CMM, RUP), productivity (e.g., CASE) and product quality (e.g., components, XP) over the past decades, chronic symptoms of cost and schedule overruns and unfulfilled user requirements are hardly cured. This paper proposed simple but robust progress monitoring systems. The progress was mathematically modeled to control both schedule and quality. That is, the progress of software development increases as the systems deliver user requirements, but it dampens with erroneous faults on the delivered systems. The development process is designed that the faults are continuously detected by the reviewers up to the industry level. Given the popularity of the UML (Unified Modeling Language) [14] and the UP (Unified Process) development [15], the progress is measured for individual use case about which all data required for progress tracking are collected. For this, we designed both a data model and critical development processes (e.g., size estimation, work order, change request) to collect the data and track progress on a near real-time basis. This serves a framework of PSS (Project Support Systems) to assist project managers to cope with schedule and quality related risky decisions. For example, managers may conduct what-if simulations on the speed of work, the change of system boundary and the quality sufficiency. As the data are recorded in a central repository over a number of projects, project managers would refer to it and adjust any potential biases of subjective decisions regarding size and efforts estimation, productivity, effect of added functionality due to change requests among others.

Our experience with the PSS at a couple of enterprise-wide projects indicated improved scheduling for the iterative development. The productivity over the earlier iterations was much deviated from what had been initially scheduled. This was mainly because the participants needed an adaptation period with unfamiliar development environments and learning appeared to take off slowly as the project proceeded. The slow learning curve of the workers at the earlier phase affected follow-up resource scheduling. Unlike the waterfall development process, the project manager also needed to cope more frequently with varying scope of iterations and to take actions for any deviations from the plan. The PSS appeared of much help to the project manager to keep under control such problems as unstable system boundary and slow learning rate. Another significant benefit with the PSS was to control schedule overruns. The system mitigated the risk considerably by allowing managers to query the

progress at a right time. The analytical reports from the progress reporting systems enabled the project manager to monitor any deviation in use case development with function points and take proper prompt actions. Any likely obstacles that hindered productivity could be identified and development staff was encouraged more toward the project goals. It should be noted that the quality dimension was also taken into account and any scheduling was performed in consideration of fault density as analyzed in a cycle where the reviewers inspected the artifacts and the developers fixed any faults found.

Not all the outcomes were affirmative. Some of the behavioral reaction to the management process was rather an opposite from what we expected. Firstly, workers did not appear to report voluntarily their daily work. They considered it as an annoying and time wasting job. So, a steady and strong guidance was required to assure them of the importance of the progress reporting. Secondly, some stakeholders and workers thought that it was an extra work to gather a collection of progress and change request related data. They appeared to object the myth that the quality assurance activities may eventually reduce downstream faults and managerial work with timely risk management, dynamic scheduling and resource planning. Thus some observational study may be required on the effect of these possibly adverse behavioral reactions. The progress monitoring systems also demand the reviewers to inspect all artifacts and play a central role to make it running. One may also question if this hard working contributes to curing the chronic symptoms of software engineering. Further research is required to minimize the manual intervention of data entry and analysis and to find effective ways of data presentation.

Acknowledgments

We wish to thank the staff of the ComponentBasis Co., Ltd. who have supported development of the system and provided the case data for the study.

References

1. The Standish Group: Chaos. The Standish Group Report (1995).
2. Albrecht, A. J.: Measuring Application Development Productivity. Proceedings of the Joint IBM/SHARE/GUIDE Application Development Symposium (1979) 83-92.
3. Daskalantonakis, M. K.: A Practical View of Software Measurement and Implementation Experience within Motorola. IEEE Transaction on Software Engineering **SE-18** (1992) 998-1010.
4. Garmus D. & Herron, D.: Function Point Analysis. Addison-Wesley (1992).
5. Herzum, P. & Sims, O.: Business Component Factory. John Wiley & Sons, Inc. (2000).
6. Jacobson, I., Christerson, M., Jonsson, P. & Overgaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992).
7. Kan, S. H.: Metric and Models in Software Quality Engineering. Addison-Wesly (2003).
8. Sprague, R. H., Jr.: A Framework for the Development of Decision Support Systems. MIS Quarterly. **4. 4** (1980) 10-26.
9. Basili, V. R. & Rombach, H. D.: The TAME Project: Toward Improvement-Oriented Software Environments. IEEE Transaction on Software Engineering **14** (6) (1988) 758-773.

10. Münch, J. & Heidrich, J.: Software Project Control Centers: Concepts and Approaches. The Journal of Systems and Software **70** (2004) 3-19.
11. Phan, D. D., George, J. F., Vogel, D. R.: Managing Software Quality in a Very Large Development Project. Information and Management **29** (1995) 277-283.
12. Antoniol, G, Fiutem, R, Lokan, C.: Object-Oriented Function Points: An Empirical Validation. Empirical Software Engineering **8** (3) SEP (2003) 225-254.
13. Tesoriero, R. & Zelkowitz, M.: Web-Based Tool for Data Analysis and Presentation. IEEE INTERNET COMPUTING **2** (5) SEP-OCT (1998) 63-69.
14. Booch, G., Jacobson, I., Rumbaugh, J.: The Unified Modeling Language User Guide. Addison-Wesley (1998).
15. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley (1999).
16. ComponentBasis R&D Center: CBPLM Management Systems ver. 3.0 User's Manual. Technical Report. ComponentBasis Co., Ltd. (2004).

Author Index